About this Help document

OpenScript Help, version 1.5

February 16, 1991

OpenScript Encyclopedia

Main Topics:

Using this Encyclopedia

Commands
Constants
Functions
Handlers/Control Structures
Messages
Operators
Parameters
Properties
Script Editor Shortcut Keys
Special Terms
Special Variables
Using DLLs with ToolBook

Glossary

© Copyright 1989-1991 by Asymetrix Corporation. All rights reserved.
ToolBook and OpenScript are registered trademarks of and DayBook is a trademark of Asymetrix Corporation.

Special thanks to Greg Riker

Adobe Type Manager® and PostScript® are registered trademarks of Adobe Systems, Inc. Corel Draw!® is a registered trademark of Corel Systems Corporation. dBase III® is a registered trademark of Ashton-Tate. Micrografx(tm), Micrografx Designer(tm), and Micrografx Draw(tm) are trademarks of Micrografx, Inc. PageMaker® is a registered trademark of Aldus Corporation. Pixie(tm) is a trademark of Zenographics, Inc. PowerPoint® is a registered trademark of Microsoft Corporation. Guide(tm) is a trademark of Owl International, Inc. Scrapbook+(tm) is a trademark of Eikon Systems, Inc.

How to use OpenScript Help

Calling up OpenScript Help

Press F1 while the active window is the ToolBook Script Editor, the ToolBook Command window, or the ToolBook Debugger.

If you press F1 while in ToolBook, and the active window is not one of those, the Help Book, and not OpenScript Help, will be opened automatically.

Context sensitive help

1. Select one or more words in the Script Editor or the Command window.

2. Press F1.

If the first word in the selection is an OpenScript keyword, OpenScript Help will open to the proper page automatically. If it is not an OpenScript keyword, OpenScript Help will open to the Index page.

Script Editor shortcut keys

Quick Reference

Keyboard shortcut keys for the script editor

Ctrl+X	Exit/Cancel
Tab	Inserts an actual tab character (ANSI 09) instead of spaces
Ctrl+Tab	Indents the selection one tab stop (formerly Ctrl+Right)
Ctrl+Left	Moves the insertion point left one word
Ctrl+Right	Moves the insertion point right one word
Ctrl+Home	Moves the insertion point to the top of the script
Ctrl+End	Moves the insertion point to the bottom of the script
Ctrl+PgUp	Moves the insertion point to the first word in the display
Ctrl+PgD	Move the insertion point to the last word in the display
Shift+Tab	Moves the selection back one tab stop (formerly Ctrl+Left)

For all Ctrl+arrow key shortcuts, you can press Shift to extend the selection. For example, Shift+Ctrl+Left extends the selection one word to the left.

Constants

Quick reference

black
blue
CR
CRLF
cyan
formfeed
green
LF
leftQuote
magenta
null
pi
quote
red
rightQuote
space
tab
white
yellow

Key constants

black

Constant

Description

The color black.

Value

The value is `0,0,0`. These three numbers represent hue, lightness, and saturation.

blue

Constant

Description

The color blue.

Value

The value is `240,50,100`. These three numbers represent hue, lightness, and saturation.

cr Constant

The carriage return constant (ANSI 13).

crlf

Constant

The carriage return/linefeed constant (ANSI 13/ANSI 10). ToolBook evaluates the `crlf` constant as two characters. All text lines in fields are terminated with the `crlf` constant.

cyan

Constant

Description

The color cyan.

Value

The value is `180,50,100`. These three numbers represent hue, lightness, and saturation.

formfeed

Constant

A character that starts a new page or a new paragraph, depending on the file format. The ANSI value of the `formfeed` constant is 12.

green

Constant

Description

The color green.

Value

The value is `120,50,100`. These three numbers represent hue, lightness, and saturation.

leftQuote

Special Term

The left quotation constant (ANSI 171).

lf

Constant

The linefeed constant (ANSI 10).

magenta Constant

Description

The color magenta.

Value

The value is `300,50,100`. These three numbers represent hue, lightness, and saturation.

null

Constant

Description

The null value.

Value

The value of the `null` constant is "" (a string with a `charCount` of `0`).

Examples

```
set text of field id 1 to null
   -- Deletes all text in field id 1

-- Prints pages where there is something in field Name
to handle print
   set printerConditions to \
      "text of field ""Name"" of this page is not null"
   start spooler
      print all pages
   end spooler
end print
```

pi

Constant

Description

The arithmetic value which is the ratio of the circumference to the diameter of a circle.

Value

The value of `pi` is `3.141592653589793`.

quote

Constant

Description

The quotation mark (").

Examples

Usually, when ToolBook evaluates a string that begins and ends with quotation marks, it strips out the quotation marks. If you want ToolBook to include quotation marks as part of a string, you can use the `quote` constant and the `&` operator, like this:

```
put "Bill" && quote & "The Whiz" & quote && "Chen" \
   into text of field "Contestant 2"
```

Another way to include a quotation mark in a string is to use two quotation marks. For example, you could write the previous statement like this:

```
put "Bill ""The Whiz"" Chen" into \
   text of field "Contestant 2"
```

red

Constant

Description

The color red.

Value

The value is `0,50,100`. These three numbers represent hue, lightness, and saturation.

rightQuote

Special Term

The right quotation mark constant (ANSI 187).

space

Constant

The space character (ANSI 32). The space character is the same character that is entered when you press the spacebar.

tab

Constant

The tab character (ANSI 9). The tab character is what is entered when you press the Tab key.

white

Constant

Description

The color white.

Value

The value is `0,100,0`. These three numbers represent hue, lightness, and saturation.

yellow

Constant

Description

The color yellow.

Value

The value is `60,50,100`. These three numbers represent hue, lightness, and saturation.

Key constants

Constant	Value	Constant	Value
keyLeftButton	1	keyRightButton	2
keyCancel	3	keyMiddleButton	4
keyBack	8	keyTab	9
keyClear	12	keyEnter	13
keyShift	16	keyControl	17
keyMenu	18	keyPause	19
keyCapital	20	keyKana	21
keyRomanji	22	keyZenkaku	23
keyHiraGana	24	keyKanji	25
keyEscape	27	keyConvert	28
keyNonConvert	29	keyAccept	30
keyModeChange	31	keySpace	32
keyPrior	33	keyNext	34
keyEnd	35	keyHome	36
keyLeftArrow	37	keyUpArrow	38
keyRightArrow	39	keyDownArrow	40
keySelect	41	keyPrint	42
keyExecute	43	keyCopy	44
keyInsert	45	keyDelete	46
keyHelp	47	key0	48
key1	49	key2	50
key3	51	key4	52
key5	53	key6	54
key7	55	key8	56
key9	57	keyA	65
keyB	66	keyC	67
keyD	68	keyE	69
keyF	70	keyG	71
keyH	72	keyI	73
keyJ	74	keyK	75
keyL	76	keyM	77
keyN	78	keyO	79
keyP	80	keyQ	81
keyR	82	keyS	83
keyT	84	keyU	85
keyV	86	keyW	87
keyX	88	keyY	89
keyZ	90	keyNumpad0	96
keyNumpad1	97	keyNumpad2	98
keyNumpad3	99	keyNumpad4	100
keyNumpad5	101	keyNumpad6	102
keyNumpad7	103	keyNumpad8	104
keyNumpad9	105	keyMultiply	106
keyAdd	107	keySeparator	108
keySubtract	109	keyDecimal	110
keyDivide	111	keyF1	112
keyF2	113	keyF3	114
keyF4	115	keyF5	116
keyF6	117	keyF7	118
keyF8	119	keyF9	120
keyF10	121	keyF11	122
keyF12	123	keyF13	124
keyF14	125	keyF15	126

keyF16	127	keyNumLock	144
keyScrollLock	145	keySemicolon	186
keyEqual	187	keyComma	188
keyDash	189	keyPoint	190
keySlash	191	keyBackQuote	192
keyLeftBracket	219	keyBackSlash	220
keyRightBracket	221	keyQuote	222

The following table describes the keyboard equivalents for some of the less obvious key constants.

Keyboard equivalents for selected key constants

Key constant	Keyboard equivalent
keyClear	5 on numeric keypad when Num Lock is off
keyMenu	Alt
keyCapital	Caps Lock (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 Num Lock is on

Glossary

Quick Reference

Application
Argument
Arithmetic expression
Arithmetic function
Author level

Book
Bug
Built-in message
Built-in property
Breakpoint

Call
Character string
Client application
Command
Command window
Comment
Constant
Container
Control structure

Debug window
Debugging
Declare
Default
Dynamic Data Exchange (DDE)
Dynamic Link Library (DLL)

Event
Execute
Execution Suspended message box
Expression

Focus
Function

Gettable property

Handler
Hierarchy

Image control
Initialize
It

Literal
Local variable
Logical expression

Message
Message handler

Message target

Nesting
Numeric value

Offscreen Image
Object
Object hierarchy
Object-oriented language
Operand
Operator

Page unit
Parameter
Persistence
Property

Reader level
Runtime error

Scope
Screen coordinate
Script
Script buffer
Script recorder
Script window
Self
Server application
Settable property
Special terms
Statement
String
String expression
String function
String specifier
Syntax
System
System book
System variable

To get handler
To set handler
ToolBook system
Trace

Unique handler identifier
User-defined function
User-defined message
User-defined property

Value
Variable
Variable window

Windows Dynamic Data Exchange (DDE)

Windows Dynamic Link Library (DLL)

Application

Definition

A computer program that performs specific tasks. A ToolBook application is one or more books that work together for a particular purpose, such as training, information management, or entertainment. ToolBook itself is an application that runs under Microsoft Windows.

Argument

Definition

See parameter .

Arithmetic expression

Definition

An expression that has arithmetic operators and numeric operands, and results in a numeric value.

Arithmetic function

Definition

A function that performs arithmetic calculations. For example, the `annuityFactor` function calculates and returns the ratio of the present value of an ordinary annuity to the annuity payment.

Author level

Definition

The working level in ToolBook that provides tools and commands for creating and modifying objects, including writing and modifying scripts for objects. Compare Reader level.

Book

Definition

A collection of ToolBook pages stored together in a file, usually based on one topic. A ToolBook application is made up of one or more books.

Bug

Definition

An error in a script. A bug might keep a script from executing, result in unpredictable behavior, or provide inaccurate results.

Built-in message

Definition

A message ToolBook automatically sends to an object when a particular event occurs. Built-in messages, which are terms in the OpenScript language, can also be sent from scripts. Compare User-defined message.

Built-in property

Definition

A property that has a corresponding term in the OpenScript language and whose possible values are determined by ToolBook. Compare User-defined property.

Breakpoint

Definition

A flag you set on a term or statement while debugging a script to indicate that you want ToolBook to pause script execution before executing the term or statement.

Call

Definition

A reference in a script to another handler.

Character string

Definition

See string.

Client application

Definition

In Windows DDE (Dynamic Data Exchange), the application that initiates the data exchange between two Windows applications. See also DDE and Server application.

Command

Definition

A term in the OpenScript language that tells ToolBook what to do. Most OpenScript commands are verbs, such as `go`, `move`, and `ask`.

Command window

Definition

A window in which you can type OpenScript statements and expressions outside of the context of a script. When you press Enter, ToolBook executes the statements or expressions.

Comment

Definition

Information in a script that explains a part of the script, but does not cause ToolBook to take any action. Comments start with two hyphens.

Constant

Definition

A specific value that can be referred to by name. For example, `pi` always refers to the value 3.141592653589793.

Container

Definition

Anything that can hold a value. The text of fields, variables, settable properties, and the Command window are containers. Container names are often used in scripts to refer to the values they hold.

Control structure

Definition

A group of statements that define the conditions under which ToolBook should perform particular actions and also define what those actions are. For example, an `if/then/else` control structure might indicate that some statements will be executed only if the value in a field is greater than 100, and other statements will be executed only if the value in that field is less than or equal to 100.

Debug window

Definition

The window in which you isolate errors in a script by tracing script execution and viewing variable values. See also Variable window.

Debugging

Definition

The process of finding errors, or bugs, in a script. You use ToolBook’s Debug window to isolate bugs in a script. After you isolate a bug, you can correct it in the Script window.

Declare

Definition

To specify whether a variable is a local or system variable, which determines its scope and persistence, and to give the variable a name.

Default

Definition

The behavior that ToolBook objects have when they are created. An object’s default behavior is determined by its type and by the default settings of its properties. You can change this behavior in a variety of ways, such as changing the values of an object’s properties, or by writing scripts to define new or different behavior for the object.

Event

Definition

An action that causes ToolBook to send a message automatically. For example, ToolBook sends the `buttonUp` message when the left mouse button is clicked. See also Message.

Execute

Definition

To carry out the actions described by the statements in a handler.

Execution Suspended message box

Definition

A message box that appears when an error occurs during script execution. This box contains the unique handler identifier for the handler that was executing when the error occurred, describes the nature of the error, and gives you the option of going to the Debug window, going to the Script window, or returning to Reader level.

Expression

Definition

Anything in an OpenScript statement that yields a value, including the name of a container, the name of a property, a literal value entered in a script, a constant, a function and its parameters, a group of operators and operands, and a string specifier.

Focus

Definition

The object that will receive the next keyboard event message at Reader level.

Function

Definition

A term in the OpenScript language that represents a particular operation that returns a value based on the parameters you give it. For example, if you type `average(sum1, sum2, sum3)` in a script, ToolBook calculates the average of the values contained in the sum1, sum2, and sum3 variables.

Gettable property

Definition

A property whose value you can find and use in a script. For example, to find out if the value of the `highlight` property is `true` or `false`, you get the value of the highlight property. All properties are gettable. See also Settable property.

Handler

Definition

A group of statements that is executed in response to a particular message or that defines a user-defined function or property. See also Message handler, To get handler, and To set handler.

Hierarchy

Definition

See Object hierarchy .

Image Control

Definition

Image control in ToolBook is a set of properties and commands that control how a page and its objects are drawn on the screen. The choice when drawing an object is whether to draw it directly to the screen or whether to compose it in another part of memory (called an offscreen image). For offscreen images, ToolBook waits until all offscreen objects are composed then transfers the resulting image to the screen.

Drawing directly to the screen allows objects to display more quickly since the step of transferring the composed image to the screen is skipped. However, drawing directly to the screen can cause some unwanted flashing of objects, especially for objects used in animations. Composing the image offscreen usually results in smoother display but can be somewhat slower.

ToolBook allows you to determine for each object whether it will be drawn offscreen or directly to the screen.

Initialize

Definition

To put a starting value into a variable.

It

Definition

A special local variable into which ToolBook puts a value if a destination for the value has not been specified, and from which ToolBook gets a value if the source of a value has not been specified.

Literal

Definition

A value entered character-by-character in a script. All literal values that contain spaces or special characters must be placed in quotation marks.

Local variable

Definition

A variable whose scope is one handler and whose persistence is the duration of the handler’s execution.

Logical expression

Definition

An expression that has logical operators and numeric, logical, or string operands, and results in a logical value.

Message

Definition

A communication sent to an object to trigger the execution of a corresponding message handler in the object’s script. If there isn’t a corresponding message handler in the object’s script, the message travels up the object hierarchy in search of one. See also Built-in message and User-defined message.

Message handler

Definition

A group of statements in an object’s script that tells ToolBook what actions to perform when the object receives a particular message. A message handler starts with to `handle`, followed by the name of the message to which it corresponds.

Message target

Definition

The object to which a particular message is first sent.

Nesting

Definition

Placing one control structure inside another control structure.

Numeric value

Definition

A value that consists of a combination of the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, E, e, +, -, $, %, and a decimal point, and that ToolBook can read as a number.

Object

Definition

A button, field, recordfield, hotword, paintObject, picture, rectangle, irregularPolygon, polygon, pie, curve, angledLine, line, arc, roundedRectangle, ellipse, group, page, background, or book. Any object can have a script. All objects have properties.

Object hierarchy

Definition

The order in which messages are passed from object to object in search of a corresponding message handler.

Object-oriented language

Definition

A programming language in which data is and behavior is encapsulated in abstract "objects." Applications are built by combining objects and specifying communication paths between them so that they may trigger each other's behavior. In ToolBook, objects are often displayed on the screen, like buttons, fields of text, and graphics.

Offscreen image

Definition

An image that is composed in memory but that is not displayed on the screen. An offscreen bitmap can be transferred rapidly to the screen using a BITBLT operation (Bit Block Transfer).

Operand

Definition

A value that an operator performs some action on. For example, in the expression totalB < totalA, the less than operator < checks to see if the value of the operand totalB is less than the operand totalA.

Operator

Definition

A symbol or a word that causes OpenScript to perform some action on values (operands), resulting in another value. For example, the + operator in the expression 7 + 3 adds the values 7 and 3 to result in a value of 10.

Page unit

Definition

The measurement used to indicate the location or movement of an object on a ToolBook page. There are 1440 page units per inch, or about 567 per centimeter. Locations are measured from the upper left corner of a ToolBook page, whether that corner is visible or not. Compare Screen coordinate.

Parameter

Definition

The part of a statement that indicates which object or other data a command is to act on; the data that a function is to use in the operation that it performs; or the data passed with a message from one handler to another. For example, parameters following a `go` command tell ToolBook what page to display.

Persistence

Definition

The conditions under which values can remain in a variable.

Property

Definition

An attribute of an object, a window, a palette, or the ToolBook system that can be modified in predetermined ways. See also Built-in message and User-defined message.

Reader level

Definition

One of two working levels in ToolBook, where you can move between pages and books, click buttons, and type text in unlocked fields. Compare Author level.

Runtime error

Definition

An error that suspends script execution and causes ToolBook to display the Execution Suspended message box.

Scope

Definition

The range of handlers that can affect or be affected by the value of a variable.

Screen coordinate

Definition

Pixels measured from the upper left corner of a monitor’s display area, used to indicate the location or movement of a ToolBook window or palette. The actual display size of a pixel depends on the type of monitor you have. Compare Page units.

Script

Definition

A series of OpenScript statements divided into parts called handlers, which describe what should happen in response to particular events. Every object, including buttons, fields, record fields, hotwords, graphics, groups, pages, backgrounds, and books, can have a script.

Script buffer

Definition

A temporary storage area where ToolBook stores the statements that result from recording actions with the script recorder. The statements remain in the script buffer until you record another series of actions or end your ToolBook session.

Script recorder

Definition

A ToolBook feature that records the results of your keystrokes and mouse actions as OpenScript statements, which can be pasted into an object’s script.

Script window

Definition

The window in which you can view, write, print, and edit an object’s script, check a script’s syntax, and paste recordings made with the script recorder.

Self

Definition

The object whose script is currently executing.

Server application

Definition

In Windows DDE, the application that responds to the Client application’s initiation of the data exchange between two Windows applications. See also Client application and Windows Dynamic Data Exchange(DDE).

Settable property

Definition

A property whose value you can define from within a script or from menus and dialog boxes in ToolBook.

Special terms

Definition

Terms in the OpenScript language other than OpenScript’s commands, functions, messages, properties, operators, and constants. Special terms include object type names, names of palettes and tools, directional words like left, right, vertical, and horizontal, and other terms used in parameters.

Statement

Definition

An instruction in a script; for example, `clear textLine 3 of text of field "x"`, or `go to next page`. Nearly all statements start with a command. The only statements that don’t are those that start and end handlers and those that start and end parts of control structures. All statements are grouped into handlers.

String

Definition

Any contiguous group of characters. All values are strings.

String expression

Definition

An expression that uses the & or && operator, has string operands, and results in a string.

String function

Definition

A function that acts on a specified string of characters. String functions do such things as convert all characters to lowercase characters or count the number of characters in a string.

String specifier

Definition

A term used to extract a string from a longer string. For example, if field "abc" contains the string `primary colors`, the expression `word 1 of text of field "abc"` returns the string `primary`. The string specifier in this expression is `word`.

Syntax

Definition

The rules for writing a script.

System

Definition

The ToolBook program. The system is at the top of the object hierarchy. System properties control attributes of ToolBook.

System book

Definition

A book whose book script can be accessed and used by many books. You can use any book as a system book.

System variable

Definition

A variable whose scope is the scripts of all objects in books opened in one instance of ToolBook. A system variable persists for one ToolBook instance.

To get handler

Definition

A handler that defines a new function or defines how to get the value of a user-defined property.

To set handler

Definition

A handler that defines the valid values for a user-defined property.

ToolBook system

Definition

See System .

Trace

Definition

The process of stepping through the execution of a script in the Debug window. You can trace statement by statement, you can trace a call to a handler, and you can trace the evaluation order of an expression.

Unique handler identifier

Definition

In the Trace box and in the Execution Suspended message box, a string that identifies the message and the object that was handling the message when execution paused.

User-defined function

Definition

A function defined by a to get handler that performs an operation and returns a value.

User-defined message

Definition

A message that you write to correspond to an event you define in a ToolBook application by writing a corresponding handler, or a message that corresponds to choosing a user-defined menu command. All user-defined messages must have corresponding handlers. Compare Built-in message .

User-defined property

Definition

A property created by an author to determine an attribute of an object in addition to the attributes determined by ToolBook's built-in properties. The author determines the name of the property, which objects have the property, and the possible values of the property. Compare Built-in property .

Value

Definition

A piece of data such as text, a number, or `true` or `false`.

Variable

Definition

A named container that is not visible in the main ToolBook window, and that maintains a value under particular conditions. These conditions vary depending on whether the variable is a local or system variable.

Variable window

Definition

The window in which you can view the values of variables while you trace script execution in the Debug window.

Windows Dynamic Data Exchange (DDE)

Definition

A communication protocol by which Windows applications can communicate with one another and share data. See also Client application and Server application .

Windows Dynamic Link Library (DLL)

Definition

A program separate from ToolBook, written in C, Pascal, or assembly language, that a book can access to extend the capabilities of the book.

Handlers and Control Structures

Quick reference

Handlers

to get
to handle
to set

Control Stuctures

conditions/when/else
do/until
if/then/else
linkDLL
start spooler
step
translateWindowMessage
while

to get

Handler structure

Syntax

```
to get <name> [<parameters>]
   <statements>
end [<name>]
```

Description

Defines a procedure for getting the value of a user-defined function or property. This is the basic structure for defining user-defined functions and for controlling what happens when a statement gets or refers to the value of a user-defined property.

Every `to get` handler must include a `return` statement with the value to be returned.

For details about using `to get` handlers, see "Defining Your Own Functions" and "Doing More with User-Defined Properties" in Chapter 6, "Beyond the Basics," in Using OpenScript. Also see the `to set` handler structure.

Parameters

The `<name>` parameter is a unique name that identifies the function or property being defined.

The `<parameters>` parameter is one or more parameters which can be passed with the function or property, separated by commas if there is more than one.

The `<statements>` parameter is one or more OpenScript statements to be executed when the object receives a `get` message containing the name specified by the `<name>` parameter.

Examples

```
to get buttonStyle
   if "button" is in target
      return the buttonStyleType of the target
   else
      set sysError to "buttonStyle target must be a button"
      return null
   end if
end buttonStyle

to get probability occurred,observations
   -- Defines a function called probability
   return occurred/observations
      -- Calculate using the arguments and return the value of
      -- the function to the calling handler
end probability

to get grossMargin sales, COGS
-- Check for valid data and return an error if data is invalid,
-- otherwise return the value of Sales - COGS
   if Sales<0 and COGS<0
```

```
      set sysError to "No valid data"
      return sysError
   end if
   if Sales<0
      set sysError to "Sales data is invalid"
      return sysError
   end if
   if COGS<0
      set sysError to "Cost data is invalid"
      return sysError
   end if
   return Sales-COGS
end grossMargin
```

to handle

Handler structure

Syntax

```
to handle <message name> [<parameters>]
   <statements>
end [<message name>]
```

Description

Executes statements when a message is passed to an object. ToolBook sends messages to indicate mouse, keyboard, and menu events, and scripts send messages using the `send` command.

This is the basic structure for a message handler, one of the building blocks of scripts. You use message handlers to define how objects respond to messages. For details about message handlers, see "Handlers: The Building Blocks of Scripts" in Chapter 2, "Script Basics," in Using OpenScript.

Parameters

The `<message name>` parameter is the name of any valid OpenScript command or message. All message names must begin with a letter and cannot include punctuation or other non-alphanumeric characters except the underscore character (_). The `<parameters>` parameter is any parameters, separated by commas, to be passed with the command or message. The `<statements>` parameter is one or more OpenScript statements to be executed when the object receives the specified message name.

Examples

```
-- Goes to the next page when the reader clicks the object
to handle buttonUp
   go to the next page
end buttonUp

-- An error handling procedure in the script for a book
to handle errorProcedure msg
   request "Error:" && msg with "Continue" or "Quit"
   if it is "Quit"
      break to system
   end if
end errorProcedure
```

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-defined property. This is the basic structure for defining what happens when statements set the value of a user-defined property. For details about using `to set` handlers with user-defined properties, see "Doing More with User-Defined Properties" in Chapter 6, "Beyond the Basics," in Using OpenScript.

Parameters

The `<name>` parameter is a unique name used to identify the property. The `<parameters>` parameter is one or more parameters that can be passed with the property, separated by commas if there is more than one. The `<value>` parameter is any expression; ToolBook sets it to the value specified in the `set` statement that calls the `to set` handler. The `<statements>` parameter is one or more OpenScript statements to be executed. To avoid creating an infinite loop, avoid referring to `<name>` in the body of the handler.

Examples

The following handler defines a `buttonStyle` property for buttons, restricts the property's value to `zoom` or `dissolve`, and sets the button's script accordingly. This handler is executed in response to statements such as `set buttonStyle of button "Next page" to "dissolve"`.

```
to set buttonStyle to effect
   -- effect is set to value passed from set statement
   conditions
      when "button" is not in target
         -- Make sure the target is a button
         set sysError to "buttonStyle target must be a button"
      when "zoom" is in effect
         -- Property's value is zoom
         set script of target to text of \
            field "zoom" of page "scripts"
      when "dissolve" is in effect
         -- Property's value is dissolve
         set script of target to text of field "dissolve" of \
            page "scripts"
      else
         set sysError to "Invalid buttonStyle"
         -- The effect is not valid
   end conditions
end buttonStyle
```

The following handler defines a `helpButton` property for buttons, sets several built-in button properties when a statement sets the property to `true` or `false`,

and prevents statements from setting the property to any other value. The handler is executed in response to statements such as `set helpButton of button "Next Page" to "false"`.

```
to set helpButton to value
   if the target contains "button"
      -- Check property is being set for a button
      conditions
         -- Set attributes for the button for each value
         when value is true
            set the fontFace of the target to "Tms Rmn"
            set the fontSize of the target to "12"
            set the borderStyle of the target to shadowed
            set the transparent of the target to true
            set the invert of the target to true
               -- Set the script for the button to forward
               -- a menu message
            set the script of the target to \
               "to handle buttonUp;forward Topics;end buttonUp"
         when value is false
            set the fontFace of the target to "system"
            set the fontSize of the target to "9"
            set the borderStyle of the target to rounded
            set the transparent of the target to false
            set the invert of the target to false
            set the script of the target to null
         else
            request "Not a valid value for this property."
      end conditions
   else
      -- Set sysError and display the message in the Command
window
      set sysError to "Property can only be set for a button."
      put sysError into the commandWindow
   end if
end helpButton
```

conditions/when/else

Control structure

Syntax

```
conditions
   [when <expression>
      [<statements>]]
   ...
   [else
      <statements>]
end [conditions]
```

Description

Executes statements in the body of the control structure based on the value of an expression. The `conditions` control structure includes zero or more `when` statements and an optional `else` statement. For details about control structures, see "Control Structures" in Chapter 2, "Script Basics," in Using OpenScript.

The `conditions` statement is evaluated by first determining the value of the expression in each `when` statement. The statements associated with the first `when` clause that evaluates to `true` are executed and control then passes to the statement following the `end [conditions]` statement. If no `when` expression is `true` and there is an `else` clause, the statements of the `else` clause are executed. If there is no `else` clause, control passes to the statement following the `end [conditions]` statement.

If you have multiple adjacent `when` statements only the last of which is followed by statements to execute, then if any of the `when` statements is true the statements following the last `when` statement are executed. This is demonstrated in the first example.

Parameters

The `<expression>` parameter is any valid expression that evaluates to `true` or `false`. The `<statements>` parameter is one or more OpenScript statements.

Examples

```
conditions

   -- Increments NorthDistrict if ZipCode is 98001, 98002,
   -- or 98003
   when ZipCode is 98001 or ZipCode is 98002 or ZipCode is 98003
      increment NorthDistrict

   -- Increments CentralDistrict if ZipCode is 98004
   when ZipCode is 98004
      increment CentralDistrict

   -- Increments SouthDistrict if ZipCode is
   -- 98005 or 98006
   when ZipCode is 98005
   when ZipCode is 98006
      increment SouthDistrict
```

```
   else
      increment Uncategorized
end conditions

to handle buttonUp loc
   system locations
   conditions
      when the target contains "button"
         forward
      when the target contains "ellipse"
         go to page "Faces" of book "c:\art\pictures.tbk"
      else
         push loc onto locations
         forward
   end conditions
end buttonUp
```

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`. 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.

Parameters

The `<statements>` parameter is one or more OpenScript statements. The `<expression>` parameter is any expression that evaluates to `true` or `false`.

Example

```
to get otherDistrictTotal
   local OtherDistricts,NumberOfDistricts
   set OtherDistricts to 0
   set NumberOfDistricts to the pageCount of this book
      -- Perform the loop for each district in NumberOfDistricts
   do
      if text of field "ZipCode" contains 98001
         increment OtherDistricts
            -- Adds 1 to OtherDistricts
      end if
      go to the next page
      decrement NumberOfDistricts
   until NumberOfDistricts = 0
   return OtherDistricts
end otherDistrictTotal
```

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>`.

Parameters

The `<expression>` parameter is any expression that evaluates to `true` or `false`. The `<statements>` parameter is one or more OpenScript statements.

Examples

```
to handle buttonUp
   if text of field id 12 contains "yes" then
      -- Displays the Sort dialog box if the value in It is "yes"
      send sort
         -- Sends the Sort menu message
   end if
end buttonUp

to handle buttonUp
   -- This handler calculates mortgage payments
   -- Declare and initialize variables
   local Rate, Price, Payment
   set Rate to 0
   set Price to 0
   set Payment to 0
      -- Get input data
   ask "What is the interest rate?"
   put it/100 into Rate
      -- Convert the interest rate
   ask "What is the price of the house?"
   put it into Price
      -- Calculate the payments and return information to reader
   put pmt(Rate, Price) into Payment
      -- User-defined function
   if Payment < 1000 then
      put Payment into text of field "Payments"
         -- Puts the value of Payment into field Payments if
         -- the value is less than 1000
   else
      if Payment <= 1500 then
         put Payment into text of field "Payments"
         request "These payments will be a stretch."
      else
         beep 5
         request "This house is too expensive!"
```

```
            -- Beeps and displays message in field if
            -- Payment exceeds 1500
      end if
   end if
end buttonUp

-- This handler evaluates student answers and increments a score
to handle answer correct_answer, student_answer
   if my score is null
      set score to 0
         -- Set score to 0 if the user-defined property
         -- has not been used
   end if
      -- If the student's answer is the same as correct_answer,
      -- increment the score
      -- and provide feedback; otherwise try again
   if the student_answer contains the correct_answer
      increment my score
      request "That's right!" with "Continue" or "Quit"
      if "Continue" is in it
         go to the next page
      else
         go to page "Menu"
      end if
   else
      -- When the student's answer is wrong
      request "Try again." with "OK" or "Review"
         -- Go to review section if student requests
      if "Review" is in it
         go to page "Review"
      end if
   end if
end answer
```

linkDLL

Control structure

Syntax

```
linkDLL <dll name>
   [<return type> <function name> ([<parameter type> \
      [,<parameter type>])]
   .
   .
   .
   [<returntype> <function name> ([<parameter type> \
      [,<parameter type>])]
end linkDLL
```

Description

Links the specified DLL so that its routines can be called from a script.

Parameters

The `<dll name>` parameter is the name of the DLL whose functions you want to make available to ToolBook and is any expression that evaluates to a valid DLL name. You can also link to a device driver by specifying the device driver's file name for the `<dll name>` parameter; be sure to include the device driver's full file name.

The `<return type>` parameter is the literal data type that the function returns. The allowed values are byte, INT, word, long, dword, float, double, STRING, or pointer. The STRING data type is returned as a string. OpenScript automatically deallocates the string after copying to free the handle.

If a DLL function returns an array, string, or data structure, follow this sequence to allocate and free memory when calling DLL functions from OpenScript:

1. Link the DLL.
2. Allocate the memory needed for the array, string, or data structure with the Windows function `globalAlloc`.
3. Lock the allocated memory with the Windows function `globalLock`.
4. Call the DLL functions from OpenScript.
5. Unlock the allocated memory with the Windows function `globalUnlock`.
6. Free the allocated memory with the Windows function `globalFree`.

The `<function name>` parameter is the literal name of the function in the DLL.

The `<parameter type>` parameter is the literal data type of the function's parameter(s). The allowed values are `BYTE`, `INT`, `WORD`, `LONG`, `DWORD`, `FLOAT`, `DOUBLE`, `STRING`, or `POINTER`.

The following table shows how the Windows data types used with DLLs compare to the standard C data types. These Windows data types are defined in the WINDOWS.H file included with the Microsoft Windows Software Development Kit.

DLL function parameter data types

DLL data type	Standard C data type
`BYTE`	unsigned char
`INT`	int
`WORD`	unsigned INT
`LONG`	Long
`DWORD`	unsigned long
`FLOAT`	float
`DOUBLE`	double
`POINTER`	void far*
`STRING`	void far*, lpstr

The `POINTER` data type is four bytes, always unsigned, and is two integers separated by a comma (segment, offset). The `POINTER` data type can be a pointer to any data type, such as a long pointer to a string or a pointer to a data structure. Although you can declare a parameter or return value as a `POINTER` data type, ToolBook will convert the parameters with the net result that all parameters are passed by value rather than reference.

You can also link by alias or by ordinal number that refers to the thunk table numeric reference of the exported function. For example, suppose you want to link the DLL named USER which contains the functions `setWindowText` and `setUpPrinter`, but you want to use another name in your scripts to call `setWindowText`. Also suppose you want to call `setUpPrinter` by its thunk table numeric reference. In this case, the `linkDLL` structure might look something like the following:

```
linkDLL user
   WORD setUpPrinter = 1(WORD)
      -- 1 is the thunk table numeric reference for
      -- setUpPrinter and can be used interchangeably in
      -- a script with the function name

   WORD setWindowTextPtr = setWindowText(WORD, POINTER)
      -- setWindowTextPtr is the alias for calling
      -- setWindowText and can be used interchangeably in a
      -- script with the function name
end linkDLL
```

Examples

```
-- Links the DLL music.dll and declares the functions
-- initializeMidi, midiPlayNote, and midiPlaySong
linkDLL "music.dll"
   WORD initializeMidi(WORD)
   WORD midiPlayNote (WORD, WORD, FLOAT)
   WORD midiPlaySong (WORD, WORD, FLOAT)
end linkDLL

linkDLL "finance.dll"
```

```
   FLOAT StdDeprec=DeprecA(WORD, WORD, FLOAT)
      -- Use the alias StdDeprec in scripts for the function
      -- DeprecA
   FLOAT GovDeprec=DeprecB(WORD, WORD, FLOAT)
      -- Use the alias GovDeprec in scripts for the function
      -- DeprecB
   FLOAT Amort(FLOAT, WORD, WORD)
   FLOAT COGS(WORD, WORD, WORD)
   FLOAT TaxRate (WORD, FLOAT, WORD)
end linkDLL
```

start spooler

Control structure

Syntax

```
start spooler
   <statements>
end [spooler]
```

Description

Starts a print job that prints the pages specified by `print` statements executed between the `start spooler` and `end [spooler]` statements. Before the statements in the body of the control structure are executed, the current printer page is set to the currently displayed page.

Using the `go` command in the body of a `start spooler` control structure causes ToolBook to navigate to the specified page but the page is not displayed. ToolBook returns to the current page when it executes the `end spooler` statement.

You cannot nest `start spooler` control structures.

Parameter

The `<statements>` parameter is any number of OpenScript statements.

Example

```
-- Prints all pages of the book if the word
-- "Complete" is in the book's caption, starting with
-- the current page
start spooler
   if the caption of this book contains "Complete"
      print all pages
   end if
end spooler
```

step

Control structure

Syntax

```
step <variable> from <start> to <finish> [by <steps>]
   <statements>
end [step]
```

Description

Allows a block of statements to be executed a specified number of times. The specified variable is set to the value of `<start>`. If the value of `<steps>` is positive, the value of the variable is incremented by that number until the value of the variable is greater than or equal to the value of `<finish>`. If the value of `<steps>` is negative, the value of the variable is decremented by the absolute value of `<steps>` until the value of the variable is less than or equal to the value of `<finish>`. The variable does not have to be declared; if it is not declared, the variable is introduced as a local variable.

When ToolBook has completely executed the `step` control structure, the value of the `<variable>` parameter is the value of the `<finish>` parameter plus the value of the `<steps>` parameter. For example, if the value of the `<finish>` parameter is `10` and the value of the `<steps>` parameter is `1`, then the value of the `<variable>` parameter will be `11` after ToolBook completely executes the control structure.

You can use the `continue` command to control execution within a `step` control structure and the `break` command to exit the `step` control structure before the variable has reached the finish value.

Parameters

The `<variable>` parameter is a variable name. The `<start>` and `<finish>` parameters are expressions that evaluate to a number. The `<steps>` parameter is an expression that evaluates to a number; its default value is `1`. The `<statements>` parameter is any OpenScript statements you want ToolBook to repeat.

The `<start>`, `<finish>`, and `<step>` parameters are evaluated only once each time ToolBook executes the `step` control structure.

Example

```
-- If the name in the variable thisPerson is in the
-- invitation list, send a printInvitation user-defined
-- message
step i from 1 to textLineCount(text of field "Invite List")
   get textLine i of text of field "Invite List"
   if it is thisPerson
      send printInvitation thisPerson
   end
end step

-- Delete pages numbered in the range specified in the
-- pageRange variable. Count down so deleting a page
-- won't effect page numbers of other pages to delete
```

```
set pageRange to "15,5"
set firstPage to min(pageRange)
set lastPage to max(pageRange)
step i from lastPage to firstPage by -1
   select page i
   send clear
end
```

translateWindowMessage

Control structure

Syntax

```
translateWindowMessage [for <winHandle>]
   before|after <winMsg> get <tbkMsg> [of <tbkObj>] return
   <dllType>
   .
   .
   .
   before|after <winMsg> send <tbkMsg> [to <tbkObj>]
end [translateWindowMessage]
```

Description

Note:

This description is for advanced users and Windows programmers.

Sets up a Windows-to-OpenScript message translation and specifies how the OpenScript message is be sent to a ToolBook message target. Zero or more clauses beginning with `before` or `after` can be specified. You must use either `before` or `after` to specify when the OpenScript message is sent in relation to the window's default processing of `<winMsg>`. The `get` form obtains a property value of the target. The `send` form specifies that a message is sent to the target object.

With the translateWindowMessage control structure, you can write scripts that allow ToolBook to take direct control of a Windows system message, whether the message is passed directly by Windows or through a DLL. The handler that translates a Windows message can specify that an OpenScript message is sent or that the value of a property is returned. The general steps to translate and handle a Windows message are:

1. Enable Windows-to-OpenScript message translation with the `translateWindowMessage` control structure.
2. Handle the OpenScript translation of the Windows message in a script using a `to handle` or `to get` handler.
3. Disable message translation with the `untranslateWindowMessage` or `untranslateAllWindowMessages` command.

To disable message translation, you can do any of the following:

- Exit ToolBook.
- Destroy a window. All translations are removed after the window receives the WM_NCDESTROY message.
- Execute the `untranslateWindowMessage` command for a specific Windows message.
- Execute the `untranslateAllWindowMessages` command for the window where you want to disable translations .

If you are writing a Windows Dynamic Link Library (DLL) see Executing OpenScript

Commands from a DLL.

Parameters

The `<winHandle>` parameter is the window handle for the window that is to translate the message. The default value is the sysWindowHandle for the ToolBook main window.

The `<winMsg>` parameter is the decimal equivalent of the hexadecimal number of the Windows message to be translated.

The `<tbkMsg>` parameter is the OpenScript form of the translated Windows message.

The `<tbkObj>` parameter is the target of the OpenScript message.

The `<dllType>` parameter specifies the data type of the return value from the `to get` handler as CHAR, BYTE, INT, WORD, LONG, DWORD, or POINTER.

Examples

```
-- This script detects WIN.INI changes and reloads values
-- that affect the book
to handle enterBook
   translateWindowMessage
      after 26 send winIniChange
   end
   -- initial loading of WIN.INI settings
end

to handle winIniChange
   -- reload related WIN.INI settings and set other
   -- properties in this book
end
```

As another example, the following translates Windows message 28 (WM_ACTIVATEAPP) into a user-defined message in OpenScript, and specifies that this OpenScript message is sent after the Windows message has finished processing.

```
translateWindowMessage
   after 28 send activateApp
end
```

A separate handler must be written to define the response for the user-defined `activateApp` message. For example:

```
to handle activateApp hwnd, winMsg, wp, lplo, lphi
   if wp=1
      -- ToolBook instance is being activated because
      -- parameter sent by Windows (wParam) is 1
      send activateInstance
   end
end
```

The parameters used in this example are:

- `hwnd` is the window handle of the window that is translating the Windows

message to a ToolBook message.

- `winMsg` is the decimal number for the Windows message that is translated to the ToolBook message activateApp.
- `wp` is the word length parameter for the Windows message.
- `lplo` is the LOWORD of the DWORD length parameter for the Windows message.
- `lphi` is the HIWORD of the DWORD length parameter for the Windows message.

The following control structure translates Windows message 19 (WM_QUERYOPEN) to an OpenScript user-defined message and specifies that this message is sent after the Windows message has finished processing.

```
translateWindowMessage
   after 19 get bookProp of book S_query return int
end
```

The `return int` keyword in this example indicates that `INT` will be the data type for the value returned by the user-defined message. The `to get` handler must always return a value appropriate for the `<dllType>` parameter specified in the `translateWindowMessage` control structure. Also, this handler must return a value appropriate for the Windows message being translated. For the message translation of 19 (WM_QUERYOPEN), the to get handler for the user-defined message bookProp must return a boolean value: non-zero to activate the instance and zero to prevent the instance from being activated.

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`. ToolBook checks the value of the expression before any of the statements are executed. Consequently, the statements in the body of the control structure are not executed if the expression evaluates to `false` when the `while` statement is first encountered.

Parameters

The `<expression>` parameter is any valid expression that evaluates to `true` or `false`. The `<statements>` parameter is an expression that evaluates to one or more OpenScript statements.

Example

```
while word 1 of text of field "ZipCode" contains "98004"
   increment SouthDistrict
   go to next page
end while
```

Messages

Quick reference

Alphabetical Index

Menu-event messages

aboutToolBook
align
author
back
background
backgroundProperties
bold
bookProperties
bringCloser
bringToFront
character
clear
command
commands
copy
createHotword
cut
drawCentered
drawDirect
exit
export
first
flipHorizontal
flipVertical
foreground
glossary
grid
group
history
import
importGraphic
index
italic
keyboard
last
new
newBackground
newPage
next
open
pageProperties
palettes
paragraph
paste
previous
printerSetup
printPages
printReport
properties
reader
redo
removeHotword
reshape
rotateLeft
rotateRight
rulers
run
save
saveAs
search
selectAll
selectPage
sendFarther
sendToBack
showHotwords
sizeToPage
sort
startRecording
stopRecording
strikeout
transparent
tutorial
underline
undo
ungroup
usingHelp

Enter-event and leave-event messages

activateInstance
enterBackground
enterBook
enterButton
enterField
enterPage
enterRecordField
enterSystem
leaveBackground
leaveBook
leaveButton
leaveField
leavePage
leaveRecordField
leaveSystem

Miscellaneous notification messages

destroy
idle
make
moved
sized
textScrolled
toggleStatus
windowMoved
windowShown
windowSized

Mouse- and keyboard-event messages

buttonDoubleClick
buttonDown
buttonStillDown
buttonUp
keyChar
keyDown
keyUp
mouseEnter
mouseLeave
rightButtonDoubleClick
rightButtonDown
rightButtonUp

DDE messages

remoteCommand
remoteGet
remoteSet

Messages: Alphabetical list

Quick reference

aboutToolBook
activateInstance
align
author
back
background
backgroundProperties
bold
bookProperties
bringCloser
bringToFront
buttonDoubleClick
buttonDown
buttonStillDown
buttonUp
character
clear
command
commands
copy
createHotword
cut
destroy
drawCentered
drawDirect
enterBackground
enterBook
enterButton
enterField
enterPage
enterRecordField
enterSystem
exit
export
first
flipHorizontal
flipVertical
foreground
glossary
grid
group
history
idle
import
importGraphic
index
italic
keyboard
keyChar
keyDown
keyUp
last
leaveBackground
leaveBook
leaveButton
leaveField
leavePage
leaveRecordField
leaveSystem
make
mouseEnter
mouseLeave
moved
new
newBackground
newPage
next
open
pageProperties
palettes
paragraph
paste
previous
printerSetup
printPages
printReport
properties
reader
redo
remoteCommand
remoteGet
remoteSet
removeHotword
reshape
rightButtonDoubleClick
rightButtonDown
rightButtonUp
rotateLeft
rotateRight
rulers
run
save
saveAs
search
selectAll
selectPage
sendFarther
sendToBack
shortcutKeys
showHotwords
sized
sizeToPage
sort
startRecording
stopRecording
strikeout
textScrolled
toggleStatus
transparent
tutorial
underline
undo
ungroup
usingHelp
windowMoved
windowShown
windowSized

aboutToolBook

Message

This message is sent to the current page when a reader or author chooses the About ToolBook command from the Help menu. ToolBook’s default response is to display the About ToolBook dialog box.

activateInstance Message

This message is sent to the current page when that instance of ToolBook becomes active. This message is sent only if the instance is at Reader level. ToolBook's default response to this message is to do nothing. If an instance becomes active because a reader clicks on the menu bar or a scroll bar, ToolBook does not send the activateInstance message until the reader releases the mouse button. If the user chooses a command from the menu bar when releasing the mouse button, ToolBook does not send the activateInstance message until after it completes the command.

align

Message

This message is sent to the current page when an author chooses the Align command from the Draw menu. ToolBook’s default response is to display the Align dialog box.

author Message

This message is sent to the current page when a reader chooses the Author command from the Edit menu. ToolBook's default response is to change the working level from Reader to Author.

Scripts that send the `author` message will cause an error in Runtime ToolBook.

back

Message

This message is sent to the current page when a reader or author chooses the Back command from the Page menu. ToolBook's default response is to display the most recent page in the session history.

background

Message

This message is sent to the current page when an author chooses the Background command from the Page menu. ToolBook's default response is to switch the focus from the foreground to the background of the current page.

If you send this message from a script, ToolBook automatically switches back to the foreground when ToolBook finishes executing the last called handler.

backgroundProperties Message

This message is sent to the current page when an author chooses the Background Properties command from the Object menu. ToolBook's default response is to display the Background Properties dialog box.

Scripts that send the `backgroundProperties` message will cause an error in Runtime ToolBook.

bold

Message

This message is sent to the current page when a reader or author chooses the Bold command from the Text menu. ToolBook's default response is to change the font style of selected text or selected field to bold. If nothing is selected, `sysFontStyle` is sent to bold and newly created fields will have a font style of bold.

bookProperties Message

This message is sent to the current page when an author chooses the Book Properties command from the Object menu. ToolBook's default response is to display the Book Properties dialog box.

Scripts that send the `bookProperties` message will cause an error in Runtime ToolBook.

bringCloser Message

This message is sent to the current page when an author chooses the Bring Closer command from the Object menu. ToolBook's default response is to bring a selected object one layer closer to the front of the foreground or background. This message causes ToolBook to change the layer number of the object. If the selected object is already on the front layer, this message has no effect.

Also see the sendFarther message.

bringToFront Message

This message is sent to the page when an author chooses Bring To Front from the Object menu. ToolBook's default response is to bring a selected object all the way to the front of the foreground or background. This message causes ToolBook to change the layer number of the object. If the selected object is already on the front layer, this message has no effect.

Also see the `sendToBack` message.

buttonDoubleClick Message

Syntax

```
buttonDoubleClick <location>, <isShift>, <isCtrl>
```

Description

This message is sent to the object whose active area contains the mouse pointer when the left mouse button is pressed and released twice within the double-click time specified in the Control Panel. ToolBook supplies three parameters that tell where the mouse button was pressed and if the Shift or Ctrl key was pressed when the button was pressed. ToolBook's default response to this message is to do nothing.

If a button is the target of a `buttonDoubleClick` message and the `highlight` property for the button is `true`, the button is highlighted before the `buttonDoubleClick` message is sent.

When a user double-clicks, the first click sends a `buttonDown` message and the second click sends a `buttonDoubleClick` message. Consequently, an object receives messages in the following order: `buttonDown`, `buttonStillDown` (if the button is held down), `buttonUp`, `buttonDoubleClick`, `buttonStillDown` (if the button is held down), `buttonUp`.

Parameters

The `<location>` parameter is a comma-separated pair of numbers indicating, in page units, where the left mouse button was pressed. The `<isShift>` and `<isCtrl>` parameters can be `true` or `false`, indicating if those keys were pressed in conjunction with the mouse button.

Example

A handler for a field:

```
to handle buttonDoubleClick arg1
   -- Select the textline that was double-clicked
   select textline (item 1 of textFromPoint(arg1) of self) \
      of the text of self
end buttonDoubleClick
```

buttonDown Message

Syntax

`buttonDown <location>, <isShift>, <isCtrl>`

Description

This message is sent to the object whose active area contains the mouse pointer when the left mouse button is pressed. ToolBook supplies three parameters that tell where the mouse button was pressed and if the Shift or Ctrl key was pressed when the button was pressed.

If a button is the target of a `buttonDown` message and the `highlight` property for the button is `true`, the button is highlighted before the `buttonDown` message is sent.

ToolBook's default response to this message is to do nothing.

Parameters

The `<location>` parameter is a comma-separated pair of numbers indicating, in page units, where the left mouse button was pressed. The `<isShift>` and `<isCtrl>` parameters can be `true` or `false`, indicating if those keys were pressed in conjunction with the mouse button.

Example

```
to handle buttonDown location, isShift
   if isShift is true
      extend select target
         -- Extend the selection to include the object that was
         -- clicked
   else
      unselect the selection
         -- Unselect selected objects
      select target
         -- Select the clicked object
   end if
end buttonDown
```

buttonStillDown Message

Syntax

`buttonStillDown <location>, <isShift>, <isCtrl>`

Description

This message is sent repeatedly to the object whose active area contained the mouse pointer when either mouse button was pressed, for as long as the mouse button remains pressed. Each time ToolBook sends this message, it updates the `<location>` parameter to identify the current location of the pointer.

The `buttonStillDown` message is useful for applications that need to track mouse drag movement, such as in layout control or animation.

ToolBook's default response to this message is to do nothing.

Parameters

The `<location>` parameter is a comma-separated pair of numbers indicating, in page units, the current location of the pointer. The `<isShift>` and `<isCtrl>` parameters can be `true` or `false`, indicating if those keys were pressed in conjunction with the mouse button.

Example

The following handler keeps track of pointer locations passed with a `buttonStillDown` message.

```
to handle buttonStillDown loc
   system Selection_coords
   local loc_count
   if loc_count is null
      set loc_count to 0
         -- Set loc_count so that it can be incremented
   end if
   push loc onto Selection_coords
      -- Use a system variable to keep a list of all
      -- locations
   increment loc_count
      -- Keep track of the number of items pushed
end buttonStillDown
```

buttonUp Message

Syntax

```
buttonUp <location>, <isShift>, <isCtrl>
```

Description

This message is sent to the object whose active area contained the mouse pointer when the left mouse button was pressed. ToolBook supplies three parameters that tell where the mouse button was pressed and if the Shift or Ctrl key was pressed when the button was pressed.

If a button is the target of a `buttonUp` message and the `highlight` property for the button is `true`, the button's highlighting is turned off before the `buttonUp` message is sent.

ToolBook's default response to this message is to do nothing.

Parameters

The `<location>` parameter is a comma-separated pair of numbers indicating, in page units, where the left mouse button was released. The `<isShift>` and `<isCtrl>` parameters can be `true` or `false`, indicating if those keys were pressed in conjunction with the mouse button.

Example

```
to handle buttonUp
   system Selection_coords
      -- Do a net selection using the first two
      -- and last two items of Selection_coords
   select all from item 1 of Selection_coords, item 2 of \
      Selection_coords to item itemCount (Selection_coords) - 1 \
      of Selection_coords, item itemCount (Selection_coords) of\
      Selection_coords
end buttonUp
```

changePrinter

Message

This message is obsolete in ToolBook version 1.5. See printerSetup.

character

Message

This message is sent to the current page when a reader or author chooses the Character command from the Text menu. ToolBook’s default response is to display the Character dialog box.

clear

Message

This message is sent to the current page when a reader or author chooses the Clear command from the Edit menu. ToolBook's default response is to delete the current selection without placing it on the Clipboard.

command Message

This message is sent to the current page when an author chooses the Command command from the Window menu. ToolBook's default response is to display the Command window.

Scripts that send the `command` message will cause an error in Runtime ToolBook.

commands

Message

The message sent to the current page when a reader or author chooses the Commands command from the Help menu. ToolBook's default response is to display a list of Reader and Author level commands in the Help book. If the Help book is not open, sending this message opens it.

copy Message

This message is sent to the current page when a reader or author chooses the Copy command from the Edit menu. ToolBook's default response is to place a copy of the current selection on the Clipboard.

createHotword Message

This message is sent to the current page when an author chooses the Create Hotword command from the Text menu. ToolBook's default response is to make the selected text into a hotword.

Also see removeHotword and showHotwords.

cut Message

This message is sent to the current page when a reader or author chooses the Cut command from the Edit menu. ToolBook's default response is to delete the current selection and place it on the Clipboard.

destroy Message

A `destroy` message is sent to an object just before it is deleted. ToolBook's default response to do nothing. This is a notification message only so writing a handler for this message cannot affect whether or not the object is deleted. To keep an object from being deleted you must write handlers for the `cut` and `clear` messages in the page, background or book.

ToolBook sends a `destroy` message to a hotword only when the ToolBook system receives a `removeHotword` message.

ToolBook does not propagate messages down the object hierarchy. Therefore, when a page receives a `destroy` message, the message is not sent to the objects on that page.

drawCentered Message

This message is sent to the current page when an author chooses the Draw Centered command from the Draw menu. ToolBook's default response is to draw an object from the center rather than from one of its corners. Draw Centered applies to lines, rectangles, rounded rectangles, arcs, ellipses, and pie wedges as well as buttons and fields.

drawDirect

Message

The menu-event message sent to the current page when an author chooses Draw Direct from the Draw menu. ToolBook's default response is to toggle the value of `sysDrawDirect` if no object is selected, or to toggle the value of `drawDirect` for a selected object.

enterBackground

Message

The message sent to the background of a page when you open a book to that page or when you go to that page from a page with a different background. ToolBook’s default response to this message is to do nothing.

enterBook Message

This message is sent to a page immediately after a book is opened. ToolBook's default response to this message is to do nothing. If a handler exists for this message, ToolBook displays the main window, executes the handler, then displays the first page. If the book is in a new instance, ToolBook instead executes the handler, displays the main window, then displays the first page.

If you use an `enterBook` handler to modify the main window, such as adjusting its position and size or changing its menu bar, you can set the `sysLockScreen` property to `true` within the handler to suppress screen updates until after ToolBook finishes executing the handler.

enterButton

Message

This message is sent to a button when it gets the focus. ToolBook’s default response to this message is to do nothing.

enterField

Message

This message is sent to a field when it gets the focus. ToolBook’s default response to this message is to do nothing.

enterPage

Message

This message is sent to a page when it gets the focus. ToolBook’s default response to this message is to do nothing.

enterRecordField

Message

This message is sent to a record field when it gets the focus. ToolBook’s default response to this message is to do nothing.

enterSystem Message

This message is sent to the current page after a book is displayed in a new instance of ToolBook. ToolBook’s default response to this message is to do nothing.

exit Message

This message is sent to the current page when a reader or author chooses the Exit command from the File menu. ToolBook's default response is to end the ToolBook session.

ToolBook does not respond to the `exit` message until ToolBook finishes executing the top-most calling handler. To exit ToolBook immediately from a script, include the statement `send exit` followed by the statement `break to system`. If you want to prevent unexpected errors from interfering, first include the statement `set sysSuspendMessages to true`.

export Message

This message is sent to the current page when a reader or author chooses the Export command from the File menu. ToolBook's default response is to display the Export dialog box.

first

Message

This message is sent to the current page when a reader or author chooses the First command from the Page menu. ToolBook’s default response is to display the first page in the current book.

flipHorizontal Message

This message is sent to the current page when an author chooses the Flip Horizontal command from the Draw menu. ToolBook's default response is to flip the selected objects horizontally.

The `flipHorizontal` message does not flip the contents of fields, bitmaps or pictures but it does flip their `bounds`.

flipVertical Message

This message is sent to the current page when an author chooses the Flip Vertical command from the Draw menu. ToolBook's default response is to flip the selected objects vertically.

The `flipVertical` message does not flip the contents of fields, bitmaps or pictures but it does flip their `bounds`.

foreground

Message

This message is sent to the current page when an author chooses the Foreground command from the Page menu. ToolBook's default response is to switch the focus from the background to the foreground of the current page.

glossary

Message

The message sent to the current page when a reader or author chooses the Glossary command from the Help menu. ToolBook's default response is to display a glossary of terms in the Help book.

grid

Message

This message is sent to the current page when an author chooses the Grid command from the Window menu. ToolBook's default response is to display the Grid dialog box.

group

Message

This message is sent to the current page when an author chooses the Group command from the Object menu. ToolBook's default response is to make two or more selected objects into a group.

Also see the `ungroup` message.

history

Message

This message is sent to the current page when a reader or an author chooses the History command from the Page menu. ToolBook's default response is to display the History dialog box. Also see the `sysHistory` property.

howToUseHelp

Message

This message is obsolete in ToolBook version 1.5. See usingHelp.

idle

Message

This message is sent to the current page when no other actions are occurring. ToolBook’s default response to this message is to do nothing.

import

Message

This message is sent to the current page when a reader or author chooses the Import command from the File menu. ToolBook's default response is to display the Import dialog box.

importGraphic Message

The message sent to the current page when an author chooses the Import Graphic command from the File menu. ToolBook's default response is to display the Import Graphic dialog box.

Also see the `ImportGraphic` command.

index

Message

The message sent to the current page when a reader or author chooses the Index command from the Help menu. ToolBook's default response is to display a list of the topics in the Help book. If the Help book is not already open, sending this message opens it.

italic Message

This message is sent to the current page when a reader or author chooses the Italic command from the Text menu. ToolBook's default response is to change the font style of selected text or a selected field to italic. If nothing is selected, `italic` will be added to the `sysFontStyle` property.

keyboard

Message

The message sent to the current page when a reader or author chooses the Keyboard command from the Help menu. ToolBook's default response is to display the Help book showing a list of keyboard equivalents for menu commands and mouse actions.

keyChar Message

Syntax

keyChar <key>,<isShift>,<isCtrl>

Description

ToolBook sends this message when a key is pressed at Reader level. ToolBook sends the message to the object with the focus or, if there is no focus, to the current page. ToolBook's 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 alphanumeric keys is to insert the corresponding character in the field.

A `keyChar` message is not sent when the user presses an accelerator key (for example, Ctrl+O for File Open) or a key that does not generate a displayable character (for example, the arrow keys and other navigation keys). This message differs from the `keyDown` and `keyUp` messages in that, for `keyChar`, the `<key>` parameter is the ANSI value of the typed character.

Write a `keyChar` handler if you want to intercept the Enter key or a printable character, such as a letter, number of punctuation character, but not including the Tab key.

Parameters

The `<key>` parameter is the ANSI value of any character. For a list of ANSI characters and their values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook.

The `<isShift>` and `<isCtrl>` parameters are `true` or `false`, indicating whether the Shift or Control keys, respectively, were pressed in conjunction with the key specified by `<key>`.

Examples

```
to handle keyChar key
   conditions
      -- Convert keyChar value to a character, then test it
   when ansiToChar(key) is in "Yy"
      -- Reader typed Y or y
      flip all pages
         -- Flip the book's pages
   when ansiToChar(key) is in "Nn"
      -- Reader typed N or n
      go to first page
         -- Go to the first page of the book
   end conditions
end keyChar

send keyChar 78
   -- Inserts "N" in the field with the focus at Reader level
```

You can put the following handler in a book script to see what parameters are returned with the `keyChar` message when a particular key is pressed:

```
to handle keyChar key
   put "- keyChar"&&key&&"("&ansiToChar(key)&")" \
      into the commandWindow
   forward
end
```

keyDown Message

Syntax

keyDown <key>,<isShift>,<isCtrl>

Description

ToolBook sends this message when a key is depressed at Reader level. ToolBook sends the message to the object with the focus or, if there is no focus, to the current page.

ToolBook's default response to this message 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, and its response for alphanumeric keys is to do nothing.

A `keyDown` message is not sent when the user presses an accelerator key (for example, Ctrl+O for File Open).

Write a `keyDown` handler if you want to intercept the Tab key or a movement-oriented key, such as the arrow keys or the PgDn key. For details about using this message and related messages, see "Controlling Keyboard Input" in Chapter 6 of Using OpenScript, "Beyond the Basics," in Using OpenScript.

Parameters

The `<key>` parameter is an integer or key constant that represents a key.

The `<isShift>` and `<isCtrl>` parameters are `true` or `false`, indicating whether the Shift or Control keys, respectively, were pressed in conjunction with the key specified by `<key>`.

See Key constants for a list of the key constants.

Examples

```
-- When the <ESC> key is pressed, save changes and exit
-- When the <PGDN> key is pressed, go to the next page
to handle keyDown key
   conditions
   when the key is keyEscape -- <ESC> key
      save changes to this book
      send exit
   when the key is keyNext   -- <PGDN> key
      go to the next page
   end conditions
end keyDown

send keyDown keyTab,false,false
   -- Move focus to next object in layer order

send keyDown keyTab,true,false
   -- Move focus to previous object in layer order

send keyDown keyTab,false,true
```

```
      -- Insert Tab character in field that has focus
```

You can put the following handler in a book script to see what parameters are returned with the `keyDown` message when a particular key is pressed:

```
to handle keyDown key, isShift, isCtrl
   put "- keyDown" && key && "(" & ansiToChar(key) &");"\
      && isShift && isCtrl into the commandWindow
   forward
end
```

keyUp Message

Syntax

keyUp <key>,<isShift>,<isCtrl>

Description

ToolBook sends this message when a key is released at Reader level. ToolBook sends the message to the object with the focus or, if there is no focus, to the current page. ToolBook's default response to this message is to do nothing.

See "Controlling Keyboard Input" in Chapter 6, "Beyond the Basics," in Using OpenScript.

Parameters

The <key> parameter is an integer or key constant that represents a key.

The <isShift> and <isCtrl> parameters are true or false, indicating whether the Shift or Control keys, respectively, were pressed in conjunction with the key specified by <key>.

See Key constants for a list of the key constants.

Examples

```
to handle keyUp key
   conditions
   when the key is keyPrior      -- When reader releases PgUp
      go to the previous page    -- Go to the previous page
   when the key is keyNext       -- When reader releases PgDn
      go to the next page        -- Go to the next page
   end conditions
end keyUp
```

You can put the following handler in a book script to see what parameters are returned with the keyUp message when a particular key is pressed:

```
to handle keyUp key, isShift, isCtrl
   put "- keyUp"&&key&&"("&ansiToChar(key)&");"&&isShift&&isCtrl\
      into the commandWindow
   forward
end keyUp
```

last

Message

This message is sent to the current page when a reader or author chooses the Last command from the Page menu. ToolBook’s default response is to display the last page of the current book.

leaveBackground

Message

This message is sent to the background when you close the book containing that background or when you go to a page with a different background. ToolBook's default response to this message is to do nothing.

leaveBook Message

This message is sent to the current page when the user quits ToolBook or goes to another book. ToolBook's default response to this message is to do nothing.

If the handler for a `leaveBook` message includes statements that make changes to a book, those changes are not saved unless the handler also includes a statement that explicitly saves changes to the book. This is also true for any other `leave` messages sent when a book is closed, including `leavePage` and `leaveBackground`.

leaveButton Message

This message is sent, at Reader level, to the button that has the focus just before another object gets the focus. ToolBook’s default response to this message is to do nothing.

leaveField

Message

This message is sent, at Reader level, to the field that has the focus just before another object gets the focus. ToolBook’s default response to this message is to do nothing.

leavePage

Message

This message is sent to the current page just before another page gets the focus. ToolBook’s default response to this message is to do nothing.

leaveRecordField

Message

This message is sent, at Reader level, to the record field that has the focus just before another object gets the focus. ToolBook’s default response to this message is to do nothing.

leaveSystem Message

This message is sent to the current page just before the instance of ToolBook is closed. ToolBook’s default response to this message is to do nothing. This is a notification message so writing a handler for this message cannot affect whether or not the instance is closed.

make

Message

This message is 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.

ToolBook’s default response to this message is to do nothing.

mouseEnter Message

This message is sent, at Reader level, to an object when the mouse pointer is moved into the object’s bounding rectangle. All objects on a page except groups can receive this message directly. ToolBook’s default response to this message depends on the object receiving the message; the mouse pointer changes to the pointer type appropriate to the object.

mouseLeave Message

This message is sent to an object when the mouse pointer is moved out of the object’s bounding rectangle. All objects on a page except groups can receive this message directly. ToolBook’s default response to this message is to do nothing.

moved

Message

The message sent, at Author level, to an object after it has been moved.

new Message

This message is sent to the current page when an author chooses the New command from the File menu. ToolBook's default response is to create a new, empty book in the active window.

newBackground Message

This message is sent to the current page when an author chooses the New Background command from the Page menu. ToolBook's default response is to create an empty page with a new, blank background. The new page is inserted after the current page. The focus is on the new page, not the new background.

newPage Message

This message is sent to the current page when a Reader or an author chooses the New Page command from the Page menu. ToolBook's default response is to create a new page with the same background as the current page. The new page is inserted after the current page.

next Message

This message is sent to the current page when a reader or author chooses the Next command from the Page menu. ToolBook's default response is to display the next page in the current book. If you send `next` when you are on the last page of a book, ToolBook displays the first page of the book.

open

Message

This message is sent to the current page when a reader or author chooses the Open Command from the File menu. ToolBook's default response is to display the Open dialog box.

To open a file without displaying the Open dialog box, see the go command.

pageProperties Message

This message is sent to the current page when an author chooses the Page Properties command from the Object menu. ToolBook's default response is to display the Page Properties dialog box so that the author can set and change the values of page properties.

Scripts that send the `pageProperties` message will cause an error in Runtime ToolBook.

palettes Message

This message is sent to the current page when an author chooses the Palettes command from the Window menu. ToolBook's default response is to display the Palettes dialog box.

Scripts that send the `palettes` message will cause an error in Runtime ToolBook.

paragraph

Message

This message is sent to the current page when a reader or author chooses the Paragraph command from the Text menu. ToolBook’s default response is to display the Paragraph dialog box.

paste Message

This message is sent to the current page when a reader or author chooses the Paste command from the Edit menu. ToolBook's default response is to paste the contents of the Clipboard into the current book.

previous Message

This message is sent to the current page when a reader or author chooses the Previous command from the Page menu. ToolBook's default response is to display the page before the current page in a book or in the Preview window. Sending the `previous` message from the first page of a book displays the last page of the book.

printerSetup

Message

The message sent to the current page when a reader or author chooses the Printer Setup command from the File menu. ToolBook's default response is to display the Printer Setup dialog box.

printPages Message

This message is sent to the current page when a reader or author chooses the Print Pages command from the File menu. ToolBook's default response is to display the Print Pages dialog box so that the reader or author can print pixel-by-pixel copies of pages from the current book.

printReport

Message

This message is sent to the current page when a reader or author chooses the Print Report command from the File menu. ToolBook’s default response is to display the Print Report dialog box so that the reader or author can print a report from the contents of specified record fields.

properties

Message

This message is sent to the current page when an author chooses the Button Properties, Field Properties, Graphic Properties, Group Properties, or Hotword Properties command from the Object menu. ToolBook's default response is to display the appropriate properties dialog box.

Scripts that send the `properties` message will cause an error in Runtime ToolBook.

quickTour

Message

This message is obsolete in ToolBook version 1.5. See tutorial.

reader

Message

This message is sent to the current page when an author chooses the Reader command from the Edit menu. ToolBook’s default response is to change the working level from Author to Reader.

redo

Message

This message is sent to the current page when a reader or author chooses the Redo command from the Edit menu. ToolBook’s default response is to reverse the effect of the Undo command.

remoteCommand Message

Syntax

`remoteCommand <command>`

Description

This message is sent to the current page when another instance of ToolBook or another Windows application sends a Windows Dynamic Data Exchange (DDE) request to execute one or more commands. For details about DDE, see "Using Windows Dynamic Data Exchange" in Chapter 6, "Beyond the Basics," in Using OpenScript.

ToolBook's default response to this message is to execute the statement `execute <command>` in the context of the currently displayed page. If the specified command contains a syntax error, or if a runtime error occurs when ToolBook tries to execute the commands, ToolBook does not process the message and instead returns a `Failed: Denied` response to the application requesting the action.

After ToolBook handles the `remoteCommand` message, ToolBook sends a `respondRemote` command to the application requesting the action. Also see the `respondRemote` command.

Parameter

The `<command>` parameter is any expression that evaluates to one or more OpenScript statements. If there is more than one statement in the list, the statements must be separated by semicolons.

Example

```
-- Intercepting remote commands with a handler in a book
-- to prevent other applications from deleting data in
-- this book
to handle remoteCommand what
   if what contains "delete" then
      -- If the remoteCommand message contains a command
      -- to delete something
      respondRemote "Failed: Denied"
         -- Send a message to the requesting
         -- application that ToolBook could not
         -- execute the command
   else
      forward
         -- Forwards the remoteCommand message
   end if
end remoteCommand
```

remoteGet Message

Syntax

remoteGet <data>

Description

This message is sent to the current page when another instance of ToolBook or another Windows application sends a Windows Dynamic Data Exchange (DDE) request for data. For details about DDE, see "Using Windows Dynamic Data Exchange" in Chapter 6 of Using OpenScript, "Beyond the Basics". Also see the getRemote command.

ToolBook's default response to this message is to execute the statement `set it to evaluate <data>` and to return the value of `It` to the requesting application. ToolBook executes this statement in the context of the currently displayed page. If there is a syntax error in the request, or if a runtime error occurs when ToolBook tries to execute the request, ToolBook does not process the message and instead returns a `Failed: Denied` response to the requesting application.

If you write a handler for a `remoteGet` message, the handler must put the requested data into `It` and must include a `respondRemote` command. If a `respondRemote` command is not executed before the end of the handler, ToolBook returns a null value to the requesting application with a `Failed: Denied` response. Also see the respondRemote command.

Parameter

The `<data>` parameter is an expression that indicates the data to be returned.

Example

```
-- Intercepting a remote request for data
to handle remoteGet what
   if what is "Your name" then
      set it to text of field "Name" of page "Personal data"
      respondRemote "OK"
   else
      forward     -- Forwards the remoteGet message
   end if
end remoteGet
```

remoteSet Message

Syntax

```
remoteSet <item>,<value>
```

Description

This message is sent to the current page when another instance of ToolBook or another Windows application sends a Windows Dynamic Data Exchange (DDE) request for ToolBook to set a property or variable to a specified value. For details about DDE, see "Using Windows Dynamic Data Exchange" in Chapter 6 of Using OpenScript, "Beyond the Basics". Also see the `setRemote` command.

ToolBook's default response to the `remoteSet` message is to execute the statement `set <item> to <value>` as though you had entered this statement in the Command window. That is, ToolBook executes this statement in the context of the currently displayed page. If there is a syntax error in the message or a runtime error when ToolBook tries to execute this statement, ToolBook sends a `Failed: Denied` response to the application requesting the data and does not process the request.

If you write a handler for a `remoteSet` message, the handler must send a `respondRemote` command. If a `respondRemote` command is not sent before the end of the handler, ToolBook sends the requesting application a `Failed: Denied` response. See `respondRemote`.

Parameters

The `<item>` and `<value>` parameters can be any expression.

Example

The following handler updates a system variable in a book when the book receives the appropriate `remoteSet` message from another book or application. The handler is in the book's script.

```
-- In the following handler, x contains the name of the
-- item to set and y contains the value.
to handle remoteSet x,y
   if x is "inflation"
      system inflation
      set inflation to y
      respondRemote OK
   end
end remoteSet
```

removeHotword Message

This message is sent to the current page when an author chooses the Remove Hotword command from the Text menu. ToolBook's default response is to turn a hotword back into ordinary text and discard its script. Also see the entries for createHotword and showHotwords.

reshape

Message

This message is sent to the current page when an author chooses the Reshape command from the Draw menu. ToolBook’s default response is to display reshape handles the author can drag to change the shape of the selected angled line, arc, curve, polygon, irregular polygon, or pie wedge.

rightButtonDoubleClick Message

Syntax

```
rightButtonDoubleClick <location>, <isShift>, <isCtrl>
```

Description

This message is sent, at Reader level, to the object whose active area contains the pointer when the right mouse button is pressed and released twice within the double-click time specified in the Control Panel.

When a user double-clicks the right 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`, `buttonStillDown` (if the button is held down), `rightButtonUp`, `rightButtonDoubleClick`, `buttonStillDown` (if the button is held down), `rightButtonUp`.

If a button is the target of a `rightButtonDoubleClick` message and the `highlight` property for the button is `true`, the button is highlighted before the `rightButtonDoubleClick` message is sent. ToolBook's default response to this message is to do nothing.

The `rightButton` messages are usually used to implement help functions.

Parameters

The `<location>` parameter is a comma-separated pair of numbers indicating, in page units, where the right mouse button was pressed. The `<isShift>` and `<isCtrl>` parameters can be `true` or `false`, indicating if one of those keys was pressed in conjunction with the mouse button.

Example

```
to handle rightButtonDoubleClick loc, isShift, isCtrl
   conditions
      when isShift is false and isCtrl is false
         go to page "Help"
      when isShift is true and isCtrl is false
         show field "Hints1"
      when isShift is false and isCtrl is true
         show field "Hints2"
      else
         forward
   end conditions
end rightButtonDoubleClick
```

rightButtonDown Message

Syntax

```
rightButtonDown <location>, <isShift>, <isCtrl>
```

Description

This message is sent, at Reader level, to the object whose active area contains the mouse pointer when the right mouse button is pressed.

If a button is the target of a `rightButtonDown` message and the `highlight` property for the button is `true`, the button is highlighted before the `rightButtonDown` message is sent. ToolBook's default response to this message is to do nothing.

Parameters

The `<location>` parameter is a comma-separated pair of numbers indicating, in page units, where the right mouse button was pressed. The `<isShift>` and `<isCtrl>` parameters can be `true` or `false`, indicating if one of those keys was pressed in conjunction with the mouse button.

Example

```
to handle rightButtonDown loc, isShift
   if isShift is true then
      select text of field "Help"
   else
      forward
   end if
end rightButtonDown
```

rightButtonUp Message

Syntax

`rightButtonUp <location>, <isShift>, <isCtrl>`

Description

This message is sent to the object whose active area contained the mouse pointer when the right mouse button was pressed. If the mouse pointer is outside the 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.

If a button is the target of a `rightButtonUp` message and the `highlight` property for the button is `true`, the button is highlighted before the `rightButtonUp` message is sent. ToolBook's default response to this message is to do nothing.

The `rightButton` messages are usually used to implement help functions.

Parameters

The `<location>` parameter is a comma-separated pair of numbers indicating, in page units, where the right mouse button was pressed. The `<isShift>` and `<isCtrl>` parameters can be `true` or `false`, indicating if one of those keys was pressed in conjunction with the mouse button.

Examples

```
to handle rightButtonUp
   if it contains "HELP"
      forward rightButtonUp
   end
end rightButtonUp

to handle rightButtonUp loc, isShift, isCtrl
   if isCtrl is true
      go to book "c:\helpbook\help1.tbk"
   else
      show field "Hints"
   end if
end rightButtonUp
```

rotateLeft Message

This message is sent to the current page when an author chooses the Rotate Left command from the Draw menu. ToolBook's default response is to rotate the selected objects 90 degrees to the left.

The `rotateLeft` message does not rotate the contents of fields, bitmaps or pictures but it does rotate their `bounds`.

rotateRight Message

This message is sent to the current page when an author chooses the Rotate Right command from the Draw menu. ToolBook's default response is to rotate the selected objects 90 degrees to the right.

The `rotateRight` message does not rotate the contents of fields, bitmaps or pictures but it does rotate their `bounds`.

rulers

Message

This message is sent to the current page when an author chooses the Rulers command from the Window menu. ToolBook's default response is to display vertical and horizontal rulers at the left and top sides of the main ToolBook window.

run

Message

This message is sent to the current page when an author chooses the Run command from the File menu. ToolBook’s default response is to display the Run dialog box.

save

Message

This message is sent to the current page when a reader or author chooses the Save command from the File menu. 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 reader or author can specify a file name for the book.

saveAs

Message

This message is sent to the current page when a reader or author chooses the Save As command from the File menu. ToolBook’s default response is to display the Save As dialog box.

search

Message

This message is sent to the current page when a reader or author chooses the Search command from the Page menu. ToolBook's default response is to display the Search dialog box so that the reader or author can search in fields for specified text.

selectAll

Message

This message is sent to the current page when a reader or author chooses the Select All command from the Edit menu. ToolBook's default response is to select all objects on the foreground or background of the current page or all text in the field the reader or author is editing.

selectPage Message

The message sent to the current page when a reader or author chooses the Select Page command from the File menu. ToolBook's default response is to select the current page. The page can be unselected using the command `clear the selection`.

sendFarther Message

This message is sent to the current page when an author chooses the Send Farther command from the Object menu. ToolBook's default response is to send the selected object one layer farther toward the back of the background or foreground from the object's original position. This message causes ToolBook to change the layer number of the selected object. If the selected object is already on the back layer, this message has no effect.

Also see the `bringCloser` message. For details about sending built-in messages, see Chapter 3, "Messages and Properties," in Using OpenScript.

sendToBack Message

This message is sent to the current page when an author chooses the Send To Back command from the Object menu. ToolBook's default response is to send the selected object all the way to the back of the foreground or background. This message causes ToolBook to change the layer number of the selected object. If the selected object is already on the back layer, this message has no effect.

Also see the `bringToFront` message.

shortcutKeys

Message

This message is obsolete in ToolBook version 1.5. See keyboard.

showHotwords Message

This message is sent to the current page when a reader or author chooses the Show Hotwords command from the Text menu. ToolBook's default response is to display rectangular outlines around hotwords throughout the current book. Also see the entries for createHotword and removeHotword.

sized

Message

The message sent to an object when its bounds or vertices properties are set at either Author or Reader level. When an author sizes an object in a selection, only that object gets the sized message, not the other selected objects. If a group is sized, the group members do not receive the sized message.

sizeToPage Message

This message is sent to the current page when a reader presses F11 or an author chooses the Size To Page command from the Window menu. ToolBook’s default response is to expand the ToolBook window to the page size of the current book or to the maximum window size, whichever is smaller.

As of ToolBook version 1.5, the ToolBook window is automatically sized to a book's page size when the book is opened. This automatic `sizeToPage` can be overridden by explicitly setting the size of the ToolBook window in an `enterBook` handler.

sort Message

This message is sent to the current page when an author chooses the Sort command from the Page menu. ToolBook's default response is to display the Sort dialog box.

Scripts that send the `sort` message will cause an error in Runtime ToolBook.

startRecording, stopRecording Message

The `startRecording` message is sent to the current page when an author chooses the Start Recording command from the Edit menu. ToolBook's default response is to start the script recorder. The `stopRecording` message is sent to the current page when an author chooses the Stop Recording command from the Edit menu. ToolBook's default response is to stop the script recorder. For details about using the script recorder, see Chapter 11, "Using the Script Recorder and the Command Window," in Using ToolBook.

Scripts that send these messages will cause an error in Runtime ToolBook.

strikeout Message

This message is sent to the current page when a reader or author chooses the Strikeout command from the Text menu. ToolBook's default response is to change the font style of selected text or a selected field to strikeout. If nothing is selected, `strikeOut` will be added to the `sysFontStyle` property.

textScrolled

Message

Syntax

```
textScrolled <scroll>
```

Description

This message is a notification message sent to a field after a reader clicks one of the scroll arrows or moves the scroll box in the scroll bar using the mouse.

This message is not sent when a field's scroll changes as a result of keyboard actions.

Parameter

`<scroll>` is the resulting value of the `scroll` property of the field.

toggleStatus Message

This message is sent to the current page when a reader or author presses F12. Toolbook’s default response is to toggle the status box on or off.

topics

Message

This message is obsolete in ToolBook version 1.5. See index.

transparent Message

This message is sent to the current page when an author chooses the Transparent command from the Draw menu. ToolBook's default response is to reverse the setting of the `transparent` property of selected objects or reverse the setting of `sysTransparent` if no objects are selected.

tutorial

Message

The message sent to the current page when a reader or author chooses the Tutorial command from the Help menu. ToolBook's default response is to start the Quick Tour.

Scripts that send the `tutorial` message will cause an error in Runtime ToolBook.

underline Message

This message is sent to the current page when a reader or author chooses the Underline command from the Text menu. ToolBook's default response is to change the font style of selected text or a selected field to underline. If nothing is selected, `underline` will be added to the `sysFontStyle` property.

undo Message

This message is sent to the current page when a reader or author chooses the Undo command from the Edit menu. ToolBook's default response is to reverse the reader's or author's most recent action, as long as the action is undoable.

When the `undo` message is sent from a script, ToolBook's default response is to do nothing.

ungroup Message

This message is sent to the current page when an author chooses the Ungroup command from the Object menu. ToolBook's default response is to separate the objects in a selected group so you can work with them individually.

Also see the `group` message and the group special term.

usingHelp

Message

The message sent to the current page when a reader or author chooses the Using Help command from the Help menu. ToolBook's default response is to display instructions for using the Help book. If the Help book is not already open, ToolBook opens it.

windowMoved

Message

The message sent to the current page when the main window has been moved. An author can write handlers for the windowMoved message to move other related windows.

windowSized

Message

The message sent to the page when the main window has been resized. An author can write handlers for the windowSized message to resize objects to match the client area size.

windowShown <isShown> Message

The message sent to the page when the main window is enlarged from or shrunk to an icon. The <isShown> parameter is true when the window is enlarged and false when it is shrunk to an icon. You can write handlers for the windowShown message to hide or show related windows.

Operators

Quick reference

&	&&	*
+	-	--
/	<	<=
<>	=	>
>=	^	argument
contains	div	is
is in	is not	is not in
mod	not	or

& Operator

Syntax

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

Description

The concatenation operator joins the expression on its left to the expression on its right.

Parameter

The `<expression>` parameter is any expression.

Example

```
put "Tool" & "Book" into it      -- Puts ToolBook into It
```

&& Operator

Syntax

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

Description

The concatenation-with-space operator joins the expression on its left to the expression on its right, with a space between them.

Parameter

The `<expression>` parameter is any expression.

Example

```
put "The" && "End" into it    -- Puts The End into It
```

* Operator

Syntax

```
<expression> * <expression>
```

Description

The multiplication operator is an arithmetic operator that multiplies the two expressions it appears between. Both expressions must yield numeric values.

Parameter

The `<expression>` parameter is any expression that yields a numeric value.

Example

```
put 25 * 4 into it      -- Puts 100 into It
```

+ Operator

Syntax

```
<expression> + <expression>
```

Description

The addition operator is an arithmetic operator that adds the two expressions it appears between. Both expressions must yield numeric values.

Parameter

The `<expression>` parameter is any expression that yields a numeric value.

Example

```
put 25.3 + 4 into it    -- Puts 29.3 into It
```

- Operator

Syntax

```
<expression> - <expression>
- <expression>
```

Description

The minus operator is an arithmetic operator that makes the expression on its right negative if it was positive or positive if it was negative or, if it is between two expressions, subtracts the expression on its right from the expression on its left. Both expressions must yield numeric values.

Parameter

The `<expression>` parameter is any expression that yields a numeric value.

Examples

```
put 25 - 4 into it          -- Puts 21 into It
put 25 - (-4) into it       -- Puts 29 into It
```

- - Comment character

Description

The comment character (two hyphens) indicates a comment. 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.

Example

```
-- This line is a comment, so ToolBook does not execute it
to handle buttonUp loc
   put loc into textLine 1       -- This comment is on a line with
a\
      of my text              -- continuation character
end buttonUp                      -- This is yet another comment
```

/ Operator

Syntax

```
<expression> / <expression>
```

Description

The division operator is an arithmetic operator that divides the expression on its left by the expression on its right, returning both the whole and fractional part of the resulting value. Both expressions must yield numeric values.

Also see the `div` operator.

Parameter

The `<expression>` parameter is any expression that yields a numeric value.

Example

```
put 25 / 4 into it      -- Puts 6.25 into It
```

< Operator

Syntax

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

Description

The less than operator is a logical operator that returns `true` if the expression on its left has a value less than the expression on its right. Otherwise, it returns `false`. The expressions can be compared as numbers, text, dates, or names.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Examples

```
put 25 < 4 into it      -- Puts false into It
if textLine 1 of text of field id 1 < textLine 1 of text of \
   field id 2 as text
   put textLine 1 of text of field id 2 into textLine 1 of \
   text of field id 1
end if
```

<= Operator

Syntax

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

Description

The less-than-or-equal-to operator is a logical operator that returns `true` if the expression on its left has a value that is less than or equal to the expression on its right. Otherwise, it returns `false`. The expressions can be compared as numbers, text, dates, or names.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Examples

```
put 25 <= 4 into it  -- Puts false into It
to handle buttonUp
   if textLine 1 of text of field "Date" <= sysDate as date
      push sysDate onto textLine 1 of text of field "Access_dates"
   end if
end buttonUp
```

<> Operator

Syntax

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

Description

The not-equal-to operator is a logical operator that yields `true` if the expression on its left and the expression on its right have different values or `false` if the expressions are equal. The expressions can be compared as numbers, text, dates, or names.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Examples

```
put 25 <> 4 into it  -- Puts true into It

if text of field "Response" <> Answer as text
   show field "Hint"
end if
```

= Operator

Syntax

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

Description

The equals operator is a logical operator that results in `true` if the expression on its left and the expression on its right have the same value. Otherwise, it returns `false`. The expressions can be compared as numbers, text, dates, or names. If the values are compared as numbers, their numeric value must be the same to result in `true`, although their formats may be different. For example, `1 = 1.000` results in `true`.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Examples

```
put 25 = 4 into it      -- Puts false into It

if text of field "Response" = Answer as text
   go to the next page
end if
```

> Operator

Syntax

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

Description

The greater than operator is a logical operator that yields `true` if the value of the expression on its left is greater than the value of the expression on its right. Otherwise, it returns `false`. The expressions can be compared as numbers, text, dates, or names.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Examples

```
put 25 > 4 into it      -- Puts true into It

if textLine 1 of text of field "City_1" > textLine 1 of text of \
   field "City_2" as text
   put textLine 1 of text of field "City_1" into textLine 1 of \
   text of field "City_2"
end if
```

>= Operator

Syntax

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

Description

The greater-than-or-equal-to operator is a logical operator that yields `true` if the expression on its left has a value that is greater than or equal to the expression on its right. Otherwise, it returns `false`. The expressions can be compared as numbers, text, dates, or names.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Examples

```
put 25 >= 4 into it  -- Puts true into It

to handle buttonUp
   if sysDate >= textLine 1 of text of field "Date" as date
      push sysDate onto textLine 1 of text of field "Access_dates"
   end if
end buttonUp
```

^

Operator

Syntax

```
<expression> ^ <expression>
```

Description

The exponentiation operator is an arithmetic operator that raises the number on its left to the power of the number on its right. Although exponentiation is evaluated from left to right, grouping is done from right to left. Therefore, `x^y^z` is evaluated as `x^(y^z)`.

Parameter

The `<expression>` parameter is any expression that yields a numeric value.

Examples

```
put 25 ^ 4 into it      -- Puts 390625 into It

put 4^1^3 into it    -- Puts 4 into It
```

and Operator

Syntax

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

Description

A logical operator that yields `true` if the expression on its left and the expression on its right are both true.

Parameter

The `<expression>` parameter is any expression.

Example

```
if text of field id 1 > 4 and text of field id 1 < 25 then
   put "The expression is true" into the commandWindow
end if
```

argument

Operator

Syntax

```
argument <expression>
```

Description

A special operator used to get the value of a parameter passed to the currently executing handler. Note that getting `argument n` and getting `item n of the argList` (where `n` is less than or equal to the value of `argCount`) can result in two different values since an argument can consist of more than one item.

Parameter

The `<expression>` parameter is any expression that evaluates to a positive integer.

Example

```
put argument 3 into text of field "Name"
   -- Puts the value of the third parameter
   -- passed to the current handler into field
   -- "Name" of the current page
```

contains Operator

Syntax

```
<expression> contains <expression>
```

Description

A logical operator that yields `true` if the expression on its right is found in the expression on its left. Otherwise, it returns `false`. This operator is not case-sensitive.

Parameter

The `<expression>` parameter is any expression.

Example

In the following example, field `What Next` contains the text `I want to study the Roman conquests`:

```
if text of field "What Next" contains "Roman" then
   -- Evaluates to true
   go to book "c:\lessons\romans.tbk"
      -- This clause is executed
else
   go to book "c:\lessons\reviews.tbk"
end if
```

div

Operator

Syntax

```
<expression> div <expression>
```

Description

An arithmetic operator that divides the number on its left by the number on its right, and yields only the integer part of the result. For an operator that yields the whole and fractional parts of a division operation, see the division operator (/) .

Parameter

The `<expression>` parameter is any expression that yields a numeric value.

Example

```
put 25 div 4.5 into it     -- Puts 5 into It
```

is Operator

Syntax

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

Description

The `is` operator is a logical operator that compares expressions and yields `true` if the expression on its left and the expression on its right are equivalent or `false` if the expressions are not equivalent. The expressions can be compared as text, numbers, dates, or names. This operator is not case-sensitive.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Example

```
to handle answer
   if Answer_text is "Julius Caesar"
      go to the next page
   else
      ask "Please try again:"
      put it into Answer_text
      send answer
   end if
end answer
```

is in

Operator

Syntax

```
<expression> is in <expression>
```

Description

A logical operator that yields `true` if the string on its left is found anywhere in the string on its right or yields `false` is the string is not found. The `is in` operator is the reverse of the `contains` operator. This operator is not case-sensitive.

Parameter

The `<expression>` parameter is any expression that yields a string.

Example

```
to handle buttonUp
   ask "Do you want to continue?"
   if "No" is in it
      break
   else
      go to page "Exam"
   end if
end buttonUp
```

is not

Operator

Syntax

```
<expression> is not <expression> [as <type>]
```

Description

The `is not` operator is a logical operator that compares expressions and yields `true` if the expression on its left and the expression on its right are not equivalent strings or `false` if the expressions are equivalent. The expressions can be compared as text, numbers, dates, or names. This operator is not case-sensitive.

Parameters

The `<expression>` parameter is any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter. The `<type>` parameter can be `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. For details about ANSI values, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property. If the `<type>` parameter is `name`, the expressions are compared as names. For details about how ToolBook compares names, see the `name` special term.

Example

```
to handle buttonUp
   ask "Who fiddled while Rome burned?"
   set answer_text to it
      -- Set the value of the variable to the text of the response
   if Answer_text is not "Nero"
      request "Please try again:"
   else
      request "Correct"
      go to the next page
   end if
end buttonUp
```

is not in

Operator

Syntax

```
<expression> is not in <expression>
```

Description

A logical operator that yields `true` if the string on its left is not found in the string on its right or `false` if the string on its left is found in the string on its right. This operator is not case-sensitive.

Parameter

The `<expression>` parameter is any expression that yields a string.

Example

```
to handle buttonUp
   ask "Who fiddled while Rome burned?"
   set answer_text to it
      -- Set the value of the variable to the text of the response
   if "Nero" is not in Answer_text
      request "Please try again:"
   else
      request "Correct"
      go to the next page
   end if
end buttonUp
```

mod

Operator

Syntax

```
<expression> mod <expression>
```

Description

An arithmetic operator that divides the number on its left by the number on its right, yielding only the remainder.

Parameter

The `<expression>` parameter is any expression that yields a numeric value.

Example

```
put 16 mod 5.0 into it     -- Puts 1 into It
```

not

Operator

Syntax

```
not <expression>
```

Description

A logical operator that yields `true` if the value of the expression on its right is false or yields `false` if the expression on its right is true.

Parameter

The `<expression>` parameter is any logical expression.

Example

```
-- If field is invisible, make it visible
if not visible of field "Hint"
   set visible of field "Hint" to true
end if
```

or

Operator

Syntax

```
<expression> or <expression>
```

Description

A logical operator that returns `true` if either the expression on its left or the expression on its right is true.

Parameter

The `<expression>` parameter is any expression.

Example

```
if Interest1 >10.5 or Interest1 < 9.5 then
   request "Interest rate is not in acceptable range."
end if
```

Commonly Used Parameters

Quick reference

The value of the parameters in the following table can be the literal values specified or they can be any expression that yields the specified values.

Parameter Descriptions

Parameter	Description
<amount>	A number.
<application>	The name of any Windows application combined with the name of any file, including their path names if necessary, and any other parameters to be passed to the application.
<book name>	The name of a book.
<character>	A character specified literally or as a constant.
<container>	(1) With `clear`, any valid container. (2) With `format`, a container name that contains the value you want to format. If `<type>` is number, contents must be numeric. (3) With `set`, any ToolBook property or other valid container. See also "Properties."
<control structure>	(1) With `exit`, must be `do`, `step`, `conditions`, or `while`. (2) With `continue`, must be `while`, `do`, or `step [<variable>]`. The `<variable>` parameter in the `next step` form of the command is the name of a step variable in the enclosing step statement and indicates the step statement on which next is to operate.
<default answer>	A string. If the text contains spaces or special terms, it must be enclosed in quotation marks.
<destination>	(1) With `pop`, the name of a container. (2) With `put`, a container.
<direction>	Must be `top`, `bottom`, `left`, or `right`.
<expression>	(1) With `beep`, a positive integer. (2) With `get` or `return`, a value. (3) With `go`, a valid identifier for a page or book. (4) With `push`, an item. (5) With `put`, a string or a number.
<fields>	If `<type>` is fixed, must be a comma-separated list of positive integers indicating the field lengths in a file to be imported or exported. If `<type>` is delimited, must be the field separator.
<file name>	A valid DOS file name or a container that contains a valid DOS file name, including the full path name if the file is not in the current directory.
<handler name>	The name of the current handler to exit from.
<level>	Must be `author`, `reader`, or `both`. The default is the current working level.

<location>	A point on the page represented by two integers separated by a comma; interpreted as page units (1440/inch) for objects or screen coordinates (pixels) for windows and palettes.
<menu item>	The name of a user-defined command as it appears on a menu, enclosed in quotation marks if the name contains spaces or punctuation.
<menu name>	The name of a menu as it appears on the menu bar, enclosed in quotation marks if the name contains spaces or punctuation.
<message>	Any valid built-in or user-defined message name. See also "Messages."
<new format>	A valid numeric, date, or time format. See also "Numeric Formats" and "Date and Time Formats."
<number of characters>	A non-negative number.
<number>	Any acceptable value. (1) With `flip` or `sort`, a positive integer. (2) With `seed`, a positive number.
<object>	(1) With `edit script`, the identifier for an object. (2) With `hide` or `show`, a hideable object, palette, or window. Enclose object names in quotation marks. (3) With `move` or `unselect`, the name, ID, or other valid identifier for an object. The default is the current selection. (4) With `send`, a ToolBook object or system. (5) With `set`, an object or window name.
<object type>	`Arc`, `angledLine`, `background`, `button`, `curve`, `ellipse`, `field`, `group`, `irregularPolygon`, `line`, `page`, `paintObject`, `pie`, `polygon`, `rectangle`, `recordField`, or `roundedRectangle`.
<objects>	A comma-separated list of object identifiers.
<old format>	A valid numeric, date, or time format. See also "Numeric Formats" and "Date and Time Formats."
<overwrite>	Must be `true` or `false`.
<pages>	With `print`, the word all or a non-negative integer indicating the number of pages to be printed, beginning with the current page.
<parameters>	One or more expressions, separated by commas if there is more than one.
<power>	Must be `1`, `2`, `4`, `8`, or `16`.
<position>	A positive integer that indicates the placement of a menu command in relation to the top of the menu.
<question>	A string with up to three lines of 50 characters each.
<reply>	A string of up to 8 characters.

<result>	`Black`, `gray`, or `white` or valid page reference.
<sort key>	A sort key consisting of `[<order>]`, `[<type>]`, `<sort expression>`, where `<order>` can be `ascending` or `descending`, `<type>` can be `text`, `number`, `date`, or `name`, and `<sort expression>` is any expression that can be evaluated in the context of a page.
<source>	One or more OpenScript statements or any container that contains OpenScript statements.
<speed>	Must be `slow`, `normal`, or `fast`.
<stack>	(1) With `pop`, the name of a container that contains a comma-separated list of items. (2) With `push`, the name of a container.
<string>	A string enclosed in quotation marks or the source of such a string.
<string specifier>	An expression that evaluates to a chunk of text in a field or record field.
<text>	A string.
<time>	A positive number.
<tool>	The field tool, button tool, record field tool, or any of the other drawing tools. See also "Drawing Tools."
<type>	(1) With `align`, must be `left`, `right`, `top`, `bottom`, `horizontal`, or `vertical`. (2) With `export` or `import`, must be fixed or delimited. (3) With `format`, must be number, date, or time. (4) With `is`, `is not`, `=`, `<>`, `=>`, `=<`, `<`, or `>`, must be `text`, `number`, `date`, or `name`.
<value>	A valid setting for a particular property.
<variable list>	One or more variable names, separated by commas if there is more than one.

(1) A number.
(2) With `move`, two numbers, separated by a comma. The first number specifies the horizontal distance and the second specifies the vertical distance to move. Distances are in page units.

Any expression that yields a number.

The name of any Windows application combined with the name of any file, including their path names if necessary, and any other parameters to be passed to the application.

Any expression that yields a number

The name of a book.

A character specified literally or as a constant.

Any expression that evaluates to one or more OpenScript statements or to a valid command string for an application.

(1) With `increment`, `clear`, or `decrement`, the name of any container.
(2) With `format`, a container name that contains the value you want to format. If `<type>` is number, contents must be numeric.
(3) With `set`, any ToolBook property or other valid container.
See also "Properties."

(1) With `break`, must be `do`, `step`, `conditions`, or `while`.
(2) With `continue`, must be `while`, `do`, or `step [<variable>]`. The `<variable>` parameter in the `step` form of the command is the name of a step variable and indicates the step statement on which `continue` is to operate.

author or reader.

Any expression that yields a value that identifies the data you want to get.

A string. If the text contains spaces or special terms, it must be enclosed in quotation marks.

the name of a container.

The name of a DLL. Must evaluate to a string.

Must be top, bottom, left, or right.

(1) With `beep`, a positive integer.
(2) With `get` or `return`, a value.
(3) With `go`, a valid identifier for a page or book.
(4) With `push`, an item.
(5) With `put`, a string or a number.
(6) With `itemCount()`, a comma separated list.
(7) With `when`, `until`, `while` or `if`, an expression evaluating to `true` or `false`.

If <type> is fixed, must be a comma-separated list of integers indicating the field lengths in a file to be imported or exported. For import, values must be positive. For export, a positive number left-justifies data in the field and a negative number right-justifies it.
If `<type>` is delimited, must be the field separator.

A valid DOS file name or a container that contains a valid DOS file name, including the full path name if the file is not in the current directory.

An expression that evaluates to a number.

The literal name of the function in a DLL.

The name of the current handler.

A list of variable names that represent values supplied
the handler is called.

Any positive integer from `1` to `32767` inclusive

True if Shift key is being pressed, false otherwise.

True if Ctrl key is being pressed, false otherwise.

Any expression that evaluates to a settable item recognized by the responding application.

A key constant or integer value that represents a key.
See key constants or keyState. For the keyChar message,
an integer ANSI value for a character.

Any expression that yields a number.

Any expression or group of expressions that yield a sequence of numbers separated by commas.

Must be `author`, `reader`, or `both`. The default is the current working level.

A point on the page represented by two integers separated by a comma; interpreted as page units (1440/inch) for objects or screen coordinates (pixels) for windows and palettes.

The name of a user-defined command as it appears on a menu, enclosed in quotation marks if the name contains spaces or punctuation.

The name of the menu item chosen.

The name of a menu as it appears on the menu bar enclosed in quotation marks if the name contains spaces or punctuation.

Any valid built-in or user-defined message name.

A valid numeric, date, or time format. See also "Numeric Formats" and "Date and Time Formats."

An expression that evaluates to the new value.

A non-negative number.

Any acceptable value.
(1) With `beep`, `flip`, `print` or `sort`, a positive integer.
(2) With `seed`, a positive number.

(1) With `edit script`, the identifier for an object.
(2) With `hide` or `show`, a hideable object, palette, or window. Enclose object names in quotation marks.
(3) With `move` or `unselect`, the name, ID, or other valid identifier for an object. The default is the current selection.
(4) With `send`, a ToolBook object or system.
(5) With `set`, an object or window name.

Arc, angledLine, background, button, curve,
ellipse, field, group, irregularPolygon, line,
page, paintObject, picture, pie, polygon,
rectangle, recordField, or roundedRectangle.

A comma-separated list of object identifiers.

An expression that evaluates to the number of bytes offset from the beginning of the data being referenced.

A valid numeric, date, or time format. See also "Numeric Formats" and "Date and Time Formats."

Must be ascending or descending. Default is ascending.

Must be `true` or `false`.

The current page.

With `print`, the word `all` or a non-negative integer indicating the number of pages to be printed, beginning with the current page.

One or more expressions, separated by commas if there is more than one.

The literal data type of the function's parameters. Allowable values are `BYTE`, `INT`, `WORD`, `LONG`, `DWORD`, `FLOAT`, `DOUBLE`, `STRING` or `POINTER`.

Any expression that yields a number. For AnnuityFactor, expressed in months.

An expression that evaluates to the pointer.

Must be `1`, `2`, `4`, `8`, or `16`.

A positive integer that indicates the placement of a menu command in relation to the top of the menu.

A string.

Any expression that yields a number. For AnnuityFactor, .10/12 = 10%

A string of up to 8 characters.

A list of nine items passed between ToolBook and an application communicating through DDE.

`Black`, `gray`, or `white` or valid page reference.

The literal data type that the function returns. Allowable values are `BYTE`, `INT`, `WORD`, `LONG`, `DWORD`, `FLOAT`, `DOUBLE`, `STRING` or `POINTER`.

Any expression that can be evaluated in the context of a page, for example, the page number or ID.

The DDE name of a Windows application.

A sort key consisting of `[<order>], [<type>], <sort expression>`, where `<order>` can be `ascending` or `descending`, `<type>` can be `text`, `number`, `date`, or `name`, and `<sort expression>` is any expression that can be evaluated in the context of a page.

One or more OpenScript statements or any container that contains OpenScript statements.

Must be `slow`, `normal`, or `fast`.

One or more OpenScript statements.

(1) With `pop`, the name of a container that contains a comma-separated list of items.
(2) With `push`, the name of a container.

An expression that evaluates to a number.

An expression that evaluates to a number. Default is `1`.

A string enclosed in quotation marks or the source of such a string.

An expression that evaluates to a chunk of text in a field or record field.

A string.

A positive number.

The field tool, button tool, record field tool, or any of the other drawing tools. See also "Drawing Tools."

Any expression that evaluates to a topic recognized
by the remote instance. The topic is usually a file name.

(1) With `align`, must be `left`, `right`, `top`, `bottom`, `horizontal`, or `vertical`.
(2) With `export` or `import`, must be fixed or delimited.
(3) With `format`, must be number, date, or time.
(4) With `is`, `is not`, `=`, `<>`, `=>`, `=<`, `<`, or `>`, must be `text`, `number`, `date`, or `name`.
(5) With `pointer` is `BYTE`, `INT`, `WORD`, `LONG`, `DWORD`, `FLOAT`, `DOUBLE`, `STRING`, or `POINTER`.

A valid setting for a particular property.

(1) With `step`, the name of the control variable for the loop. The control variable will be incremented each cycle through the loop.
(2) With `continue step`, the control variable on which `continue` is to operate. If `variable` is omitted, the innermost `step` control structure is assumed.

One or more variable names, separated by commas if there is more than one.

The window handle of a window (ex. sysWindowHandle).

The decimal equivalent of the hexadecimal number of a Windows message.

Properties

Quick reference

Alphabetical List

Book Properties

backgroundCount
caption
captionShown
customColors
footer
header
icon
name
object
pageCount
script
size
uniqueName
userProperties

Background properties

fillColor
idNumber
name
object
objects
parent
pattern
script
storedImages
pageCountstoreImage
strokeColor
uniqueName
userProperties

Page properties

idNumber
imageInvalid
name
object
objects
pageNumber
parent
script
storedImages
storeImage
uniqueName
userProperties

Group properties

bounds
drawDirect
idNumber
layer
name
object
objects
parent
position
script
uniqueName
userProperties
vertices
visible

Button properties

borderStyle
bounds
caption
checked
drawDirect
excludeTab
fillColor
fontFace
fontSize
fontStyle
highlight
idNumber
invert
layer
name
object
parent
position
script
strokeColor
textOverflow
transparent
uniqueName
userProperties
vertices
visible

Field and record field properties

activated
baselines
borderStyle
bounds
drawDirect
drawTextDirect
fieldType
fillColor
fontFace
fontSize
indents
layer
name
object
objects
parent
position
script
scroll
selectedTextLines
tabSpacing
tabType
text
textAlignment
textOverflow
transparent
uniqueName
userProperties
vertices
visible

fontStyle
idNumber
spacing
strokeColor

Draw object, paint object, and picture properties

bounds
drawDirect
fillColor
idNumber
layer
lineStyle
name
object
parent
pattern
position
script
strokeColor
transparent
uniqueName
userProperties
vertices
visible

Window and palette properties

bounds
object
position
vertices
visible

ToolBook regular system properties

focus
selectedText
selectedTextState
selection
self
sysAlignment
sysBooks
sysCentered
sysChangesDB
sysClientHandle
sysCursor
sysDate
sysDateFormat
sysDrawDirect
sysError
sysErrorNumber
sysFillColor
sysFontFace
sysFontSize
sysFontStyle
sysGrid
sysGridSnap
sysGridSpacing
sysHistory
sysHistoryRecord
sysHotwordsShown
sysIndents
sysLevel
sysLineSpacing
sysLineStyle
sysLockScreen
sysMagnification
sysMousePosition
sysNumberFormat
sysOperatingSystem
sysPageScroll
sysPasswords
sysPattern
sysPolygonShape
sysRuler
sysRuntime
sysStrokeColor
sysSuspend
sysSuspendMessages
sysTabSpacing
sysTabType
sysTime
sysTimeChar
sysTimeFormat
sysTransparent
sysUnits
sysVersion
sysWindowHandle
target

ToolBook printer properties

printerArrangement
printerBorders
printerBottomMargin
printerClipText
printerConditions
printerFieldNames
printerFields
printerFieldWidths
printerGroupsAcross
printerGutterHeight
printerGutters
printerGutterWidth
printerLabelWidth
printerLeftMargin
printerMargins
printerName
printerPageBitmap
printerRightMargin
printerSize
printerStyle
printerTopMargin

ToolBook international system properties

sysCountry
sysCurrency
sysDecimal
sysErrorNumber
sysEvening
sysICountry
sysICurrDigits
sysICurrency
sysIDate
sysIDigits
sysILZero
sysIMeasure
sysINegCurr
sysITime
sysITLZero
sysLanguage
sysList
sysLongDate
sysMorning
sysShortDate
sysThousand
sysTimeChar

ToolBook startup system properties

startupBook
startupDrawDirect
startupHeight
startupSysBooks
startupSysColors
startupUnits
startupWidth

Properties - Alphabetical List

Quick reference

activated
backgroundCount
baselines
borderStyle
bounds
caption
captionShown
checked
customColors
drawDirect
drawtextdirect
excludeTab
fieldType
fillColor
focus
fontFace
fontSize
fontStyle
footer
header
highlight
icon
idNumber
imageInvalid
indents
invert
layer
lineStyle
name
object
objects
pageCount
pageNumber
parent
pattern
position
printerArrangement
printerBorders
printerBottomMargin
printerClipText
printerConditions
printerFieldNames
printerFields
printerFieldWidths
printerGroupsAcross
printerGutterHeight
printerGutters
printerGutterWidth
printerLabelWidth
printerLeftMargin
printerMargins
printerName
printerPageBitmap
printerRightMargin
printerSize
printerStyle
printerTopMargin
script
scroll
selectedText
selectedTextLines
selectedTextState
selection
self
size
spacing
startupBook
startupDrawDirect
startupHeight
startupSysBooks
startupSysColors
startupUnits
startupWidth
storedImages
storeImage
strokeColor
sysAlignment
sysBooks
sysCentered
sysChangesDB
sysClientHandle
sysCountry
sysCurrency
sysCursor
sysDate
sysDateFormat
sysDecimal
sysError
sysErrorNumber
sysEvening
sysFillColor
sysFontFace
sysFontSize
sysFontStyle
sysGrid
sysGridSnap
sysGridSpacing
sysHistory
sysHistoryRecord
sysHotwordsShown
sysICountry
sysICurrDigits
sysICurrency
sysIDate
sysIDigits
sysILZero
sysIMeasure
sysINegCurr
sysIndents
sysITime
sysITLZero
sysLanguage
sysLevel
sysLineSpacing
sysLineStyle
sysList
sysLockScreen
sysLongDate
sysMagnification
sysMorning
sysMousePosition
sysNumberFormat
sysOperatingSystem
sysPageScroll
sysPasswords
sysPattern
sysPolygonShape
sysRuler
sysRuntime
sysShortDate
sysStrokeColor
sysSuspend
sysSuspendMessages
sysTabSpacing
sysTabType
sysThousand
sysTime
sysTimeFormat
sysTransparent
sysUnits
sysVersion
sysVersion
sysWindowHandle
tabSpacing
tabType
target
text
textAlignment
textOverflow
transparent
uniqueName
userProperties
vertices
visible

activated

Property

Description

A foreground field and record field property used to set or get whether a field's script and hotwords are active. For background fields, this property is only gettable because fields on a background are always activated. Setting the `activated` property to `true` is equivalent to checking Activate Scripts in the Field Properties dialog box. If `activated` is set to `true` for a particular field, the contents of that field cannot be edited by a reader, the mouse pointer does not change to the text insertion point or arrow when it enters the protected field, the field's hotwords are active, and clicking on the field sends button messages to the field.

Value

The value can be `true` or `false`. The default value is `false` for all record fields and fields on a foreground. Background fields are always activated so the value of `activated` is `true`.

backgroundCount Property

Description

A book property used to get the number of backgrounds in a book.

Value

The value is a positive integer indicating the number of backgrounds in a book.

baselines

Property

Description

A field and record field property used to set or get whether the text baselines in a field are visible. Setting the baselines property to `true` is the equivalent of checking Baselines in the Field Properties dialog box.

Value

The value can be `true` or `false`. The default value is `false`.

borderStyle Property

Description

A button, field, or record field property used to set or get button border styles or field border styles.

Value

The allowable values for button border styles are `none`, `pushbutton`, `rectangle`, `rounded`, `shadowed`, `radiobutton`, and `checkbox`. The default value for button border styles is `pushbutton`. The allowable values for field border styles are `none`, `rectangle`, `scrolling`, and `shadowed`. The default value for field border styles is `rectangle`.

Example

```
set borderStyle of button "Editor" to checkbox
   -- Sets the border style for a button
```

bounds Property

Description

A field, button, group, graphic, or window property used to set or get the location and size of the bounding rectangle of a field, a button, a group, a graphic, or the ToolBook main window. This property can also be used to get the coordinates of the bounding rectangle for the Command window and for palettes. For graphic objects, the bounds of the object are analogous to the location of the selection handles of the object as shown at Author level.

Scripts that get the bounds of palettes or the Command window will cause an error in Runtime ToolBook.

Value

The value is four comma-separated numbers that together define the bounding rectangle of the object (for example, 5025, 1640, 7780, 2880). The first two numbers give the x and y coordinates of the upper left corner of the bounding rectangle, and the last two numbers give the x and y coordinates of the lower right corner. For fields, buttons groups, and graphic objects, the numbers are in page units. For windows and palettes, the numbers are in screen coordinates (pixels).

caption Property

Description

A book property used to set or get text shown in the caption bar instead of the book name, and a button property used to set or get the text in a button’s label.

Setting the `caption` property for books is equivalent to entering a value in the Caption box in the Book Properties dialog box. Setting the `caption` property for buttons is equivalent to entering a value in the Button Label box in the Button Properties dialog box.

Value

The value is a string that is the book caption or button label as shown in the Book Properties or Button Properties dialog box. The string can be up to 32 characters long for book captions and up to 255 characters long for button labels. The default value for books is `null`. The default value for buttons is `Button`.

captionShown Property

Description

A book property used to set or get whether the book caption is shown in the caption bar. Setting the `captionShown` property to `true` is the equivalent of checking Show Caption in the Book Properties dialog box.

Value

The value can be `true` or `false`. The default value is `false`.

checked

Property

Description

A button property used to set or get whether a checkbox-style button is checked or a radio-style button is filled in.

Value

The value can be `true` or `false`. The default value is `false`.

customColors Property

Description

A book property used to set or get the 64 colors of the color tray.

The 64 default custom colors are the colors of the color tray when you choose the color tray's Revert button.

Value

The value consists of 64 text lines; each text line is a comma-separated list of three numbers that represents the hue, lightness, and saturation of a single color in the color tray. Each text line is terminated with `crlf`. The colors in the color tray are numbered from top to bottom, left to right.

The hue value ranges from 0 to 360, corresponding to the angle in a color wheel where:

`0` is red	`60` is yellow	`120` is green
`180` is cyan	`240` is blue	`300` is magenta

Intermediate angles represent intermediate colors. `0` and `360` are equivalent.

The lightness value ranges from 0 to 100, with 0 representing 0% lightness (black) and 100 representing 100% lightness (white).

The saturation value ranges from `0` to `100`, with `0` representing 0% saturation (gray) and `100` representing 100% (pure color).

ToolBook may have to convert the value you specify to the closest value that represents an available color. The default value is the value of the current colors in the color tray.

Examples

```
set textline 32 of customColors of this book to 0,0,0
   -- Sets the 32nd color tile in the color tray
   -- palette to black

set item 1 of textline 3 of customColors of this book to 180

to handle enterBook
   -- Put the customColors into the user-defined property
   -- originalColors
   set originalColors of this book to customColors of this book
end enterBook

to handle leaveBook
   -- Restore the original colors (in case user has
   -- changed them)
   set customColors of this book to originalColors of this book
end leaveBook
```

drawDirect Property

Description

An object property that specifies the method for drawing the screen image of the object. All object types except hotwords, pages, and books have this property.

Value

The value can be `true` or `false`; the default is the value of `sysDrawDirect`. If `drawDirect` is true, the object is drawn directly on screen; if `false`, it is drawn as part of the offscreen image.

Objects with this property set to `true` will usually display more quickly than objects with this property set to `false`. For details about other techniques to optimize page display speed, see Chapter 3, "Tips for ToolBook Authors," in the ToolBook Ideas booklet.

Example

```
set the drawDirect of button "Example" to true
   -- draws button "Example" directly to the screen when the
   -- page is displayed or the button is moved or sized.
```

drawTextDirect Property

Description

A field or record field property that specifies the method for drawing the screen image of text in a field. A field's or record field's borders, including scroll bars, are drawn as specified by the `drawDirect` property.

Value

The value can be `true` or `false`; the default is `false` when `drawDirect` is `false` for the field. If `drawTextDirect` is `true`, the text is drawn directly on screen even if the field is drawn in the offscreen image; if `false`, the text is drawn as part of the offscreen image when the field is drawn offscreen. Setting this property to `true` is the same as checking Draw Text Direct in the Field Properties or Record Field Properties dialog box.

Example

```
set the drawTextDirect of recordField "Instructions" to true
```

excludeTab Property

Description

A button property that specifies whether or not the button is included in the tabbing order.

Value

The values are `true` or `false`. The defalult is `false`. When excludeTab is `true`, the button cannot get the focus by tabbing, and therefore the button's script cannot run as a result of keyboard action.

fieldType Property

Description

A field or record field property that determines the field's behavior. This property effects the type of word wrap applied in displaying text in a field. It also determines whether the field is a single-select or multi-select listbox.

Value

The fieldType property can have one of the following values:

- `wordWrap` Allows normal word wrap at the right margin when a user types or pastes text in the field. Use this style to display text or to allow readers to enter text. Set the `activated` property of the field to true if you want readers to be able to click hotwords in this type of field of if you want to lock the field so that readers cannot edit text.
- `noWrap` Prevents word wrap at the right margin for each paragraph. That is, individual lines of text are created only by pressing Enter while typing or at every CRLF in pasted or imported text. You cannot scroll the insertion point past the right edge of the field, even if a line of text continues past the border.
- `singleLineWrap` Allows users to enter a single line of text between the field's borders. Pressing Enter or Shift+Enter has no effect on line breaks in this type of field. Use this type of field to create data entry forms, and set the `activated` property of the field to true if you want readers to be able to click hotwords in this type of field.
- `singleSelect` Allows readers to move the focus in the list and to select a text line, as in a Windows-style listbox. Each paragraph appears as a single text line, with no word wrap at the right margin, and paragraph alignment options have no effect. Scripts are automatically activated for this type of field. (That is, readers cannot enter text.)
- `multiSelect` Allows readers to move the focus in the list and to select multiple, discontinuous text lines, as in a multiple-choice Windows style listbox. Other behavior is the same as a single-select listbox.

For listbox fields (the `fieldType` property set to `singleSelect` or `multiSelect`) keyboard and mouse actions produce these results:

- Pressing a key selects a textline that begins with that letter, and repeatedly pressing a key cycles through all textlines that begin with that letter.
- Dragging the mouse or pressing an arrow key moves the focus, which appears as a dotted rectangle enclosing the textline. For single-select listbox fields, moving the focus also moves the selection to the textline that has the focus. For multi-select listbox fields, clicking a textline or pressing the space bar toggles selection of the textline.

Listbox fields in ToolBook are similar to Windows 3.0 listboxes in most ways, except:

- Text can be edited directly at Author level, rather than writing a program to set the contents.

- A listbox field can contain hotwords, the text can be formatted in multiple fonts, baselines can be shown, and all types of paragraph formatting except alignment can be displayed.
- The textlines are not sorted automatically in a ToolBook listbox.

For information about writing scripts for listboxes, see the `selectedTextLines` field property.

fillColor Property

Description

A button, field, record field, draw object, paint object, or background property used to set or get the fill color of an object.

Value

The value is a comma-separated list of three non-negative numbers representing the hue, lightness, and saturation of a single color.

The hue value ranges from 0 to 360, corresponding to the angle in a color wheel where:

```
0 is red            60 is yellow        120 is green
180 is cyan         240 is blue         300 is magenta
```

Intermediate angles represent intermediate colors. 0 and 360 are equivalent.

The lightness value ranges from 0 to 100, with 0 representing 0% lightness (black) and 100 representing 100% lightness (white).

The saturation value ranges from 0 to 100, with 0 representing 0% saturation (gray) and 100 representing 100% (pure color).

ToolBook may have to convert the value you provide to the closest value that represents an available color. The default value is the value of the `sysFillColor` property.

focus Property

Description

A system property used to set or get the object on the current page or background that has the focus.

If a button has the focus, its caption will be surrounded by a focus outline (a dashed rectangle). If the button's caption is null then no outline will appear but the button will still have the focus. If a button has its `excludeTab` property set to `true` it cannot receive the focus through a keyboard action (like tabbing) but will receive the focus when it is clicked with the mouse.

Value

The value is the uniqueName of the object on the current page or background that will receive keyboard events at Reader level, or null if there is no focus.

fontFace Property

Description

A button, field, or record field property used to set or get the default font for button labels and field text.

If this property is set to a value that specifies a font typeface not available in the Character dialog box, the font typeface in effect before the `fontFace` property is set remains the object's default font typeface. If, at a later time, the font typeface specified as the value for the property becomes available, that font typeface is used.

Value

The value is an expression that evaluates to the name of the default font for a button label or field text as shown in the Character dialog box for a button label or field text. The default value is the value of the `sysFontFace` property.

Example

```
set the fontFace of button "Next" to "Tms Rmn"
   -- Changes the font for the button label to Tms Rmn
```

fontSize

Property

Description

A button, field, or record field property used to set or get the default point size of button labels or field text.

If this property is set to a value that specifies a point size not available for the object’s default font face, the nearest smaller valid value is used.

Value

The value is an expression that evaluates to a positive integer, which defines the point size of the default font as shown in the Character dialog box for a button or field. The default value is the value of the `sysFontSize` property.

fontStyle Property

Description

A button, field, or record field property used to set or get the font style of button labels or field text.

Value

The value is an expression that evaluates to null or to a comma-separated list of one or more of the values `bold`, `italic`, `underline`, and `strikeout`. The combination of these values describes the style of the object's default font. A null value is equivalent to plain style. The default value is the value of the `sysFontStyle` property.

footer

Property

Description

A book property used to set or get the contents of a footer to be printed on each printed page.

Value

The value is an expression that evaluates to a string of up to 32,000 characters. The default value is null, or no footer. You can include three special codes in a footer to print out the date, time and page number. They are:

~d date

~t time

~p page number

For example, you can set the footer:

```
set footer of this book to "Date: ~d  ~t" & CRLF & "Page ~p"
```

Any printed pages would have the date on the first line and the page number on the second line of the footer.

header Property

Description

A book property used to set or get the contents of the header to be printed on each printed pages.

Value

The value can be any expression that evaluates to a string of up to 32,000 characters. The default value is null, or no header. You can include three special codes in a header to print out the date, time and page number. They are:

~d date

~t time

~p page number

For example, you can set the header to:

```
set header of this book to "Date: ~d  ~t"
```

Any printed pages would have the date and time in the header.

highlight

Property

Description

A button or hotword property used to set or get whether a button or hotword is highlighted briefly when it is clicked. Setting the `highlight` property to `true` is the equivalent of checking Highlight in the Button or Hotword Properties dialog box.

Value

The value can be `true` or `false`. The default value is `true` for buttons and `false` for hotwords.

icon Property

Description

A book property that sets or gets the icon displayed when a book is minimized. The `icon` property can be set to the icon of another book or to the name of a icon resource file (.ico file).

The `icon` property does not effect the icon displayed by the Program Manager. To set the icon displayed by the Program Manager use the Properties command from the Program Manager's File menu and type the name of the icon file containing the icon.

Icon files can be created using tools like SDKPaint supplied with the Microsoft Windows Software Developement Kit.

Value

The icon property can be set to the value of the `icon` property of another book or to the name of an icon file. Once the icon property is set, the icon file is no longer required.

If you display the value of the `icon` property (such as putting it into the Command window), the name of the resource file used to create the icon will be displayed.

If you set the `icon` property to the name of a file that isn't a proper .ico file or a file that doesn't exist `sysError` will be set to "Specified file is not a valid icon" and `sysErrorNumber` will be set to 8214.

Example

```
set the icon of this book to the icon of book "clipart.tbk"

set the icon of this book to "logo.ico"

set the icon of this book to "umbrella.ico"
set the text of field "icon name" to the icon of this book
   -- Places the text "umbrella.ico" into the text of
   -- the field "icon name" on this page.
```

idNumber Property

Description

A background, page, button, field, hotword, group, or graphic object property used to get the ID number of an object.

Value

The value is a positive integer that is unique to each page in a book, each background in a book, and each object on a page or background. The value can be in the range 0 through (2^32 - 1).

imageInvalid Property

Description

A field or record field property that indicates an image has been changed since it was last stored. Images are stored using the `store` command or by setting the `storeImage` property to `true`.

Value

The value can be `true` or `false`. If `true`, the page or background has changed since the image was stored. This property cannot be set. 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.

indents

Property

Description

A field and record field property used to set or get the width of paragraph indents for a field.

Value

The value is three non-negative numbers separated by commas. The first number is the first line indent, the second is the left indent for a paragraph, and the last is the right indent for a paragraph. The numbers are interpreted in page units. The default value is `0,0,0`.

invert

Property

Description

A button or hotword property used to set or get whether the color of a button or hotword is inverted. If the `invert` property is set to `true`, the stroke and fill colors of the button or hotword are inverted.

Value

The value can be `true` or `false`. The default value is `false`.

layer

Property

Description

A button, field, group, or graphic object property used to set or get the layer number of a button, field, or graphic object on a page. Setting this property is equivalent to entering a value in the Layer box in the Properties dialog box for the object.

Value

The value is a positive integer identifying the current layer number of the object on the page or background.

lineStyle

Property

Description

A graphic object property used to set or get the style of lines drawn with the line or shape tools and the line drawn around the border of paint objects and imported pictures.

Value

The value can be `none`, `0`, `1`, `2`, `3`, `4`, `6`, or `8` to represent the number of pixels in the line width, `dashed` for a dashed line style, or `dotted` for a dotted line style. The default value is the value of the `sysLineStyle` property when the object was created.

name

Property

Description

A button, field, group, page, background, hotword, or graphic object property used to set or get the name of the object as it appears in the Properties dialog box for the object. Also a book property used to get the file name of a book.

Value

For buttons, fields, groups, pages, backgrounds, and graphic objects, the value is a string of up to 32 characters. For books, the value is the valid path name of the book file. The default value is null.

object

Property

Description

An object property used to get the type of object.

Value

This property may have any one of these values:

angledline	field	picture
arc	group	pie
background	irregularpolygon	polygon
book	line	recordfield
button	page	rectangle
curve	paintobject	roundedrectangle
ellipse		

Note that all values are returned in lowercase.

objects

Property

Description

A field, record field, group, page, or background property used to get the uniqueNames of hotwords in a field, the uniqueNames of objects in a group, or the uniqueNames of buttons, fields, and graphic objects on a page or background.

The `objects` property of a page does not include the uniqueNames of record fields. This is because record fields exist on the background, not on the page.

Value

The value is a comma-separated list of unique object names.

pageCount

Property

Description

A book or background property used to get the number of pages in a book or sharing a background. You must specify the book or background.

Value

The value is a positive integer indicating the number of pages in a book or sharing a background.

Examples

```
get the pageCount of this background
   -- Returns the number of pages sharing the current
   -- background

get the pageCount of this book

get the pageCount of book "lessons.tbk"
```

pageNumber Property

Description

A page property used to set or get the page number of a page in a book.
Changing the value of this property changes the order of pages in a book.

Value

The value is a positive integer that gives the page’s ordinal position in its book.

parent

Property

Description

A button, hotword, field, record field, group, graphic, page, or background property used to get the parent of an object. The parent of an object is the object immediately above it in the object hierarchy. For example, the parent of a hotword is always a field. For an object in a group, the parent object is the group. If an object is not a hotword and is not in a group, its parent object is the page or background on which it lives. For a page, the parent object is a background. For a background, the parent object is a book.

Value

The value is the uniqueName of the parent object.

pattern

Property

Description

A graphic or background property used to set or get the current fill pattern that is used in draw objects, or the pattern that the background is filled with.

If this property is set while draw objects are selected, the fill patterns of the selected draw objects are changed.

Value

The value must be `none`, `solidStroke`, `solidFill`, or an integer in the range from `1` through `128` or `253` through `255`. For the values `253` through `255`, `253 = none`, `254 = solidFill`, and `255 = solidStroke`. The default value is the value of the `sysPattern` property.

position

Property

Description

A property of a field, button, graphic, group, palette, or window used to set or get its location.

Value

The value is a comma-separated pair of numbers that together define the location of the upper left corner of the object. For windows, the coordinates are measured in screen coordinates from the top left corner of the screen. For fields, buttons, or graphic objects, the coordinates are measured in page units relative to the top left corner of the page.

Scripts that get or set the position of palettes or the Command window will cause errors in Runtime ToolBook.

printerArrangement Property

Description

A system property used to set or get the number of ToolBook pages printed per sheet.

Value

The value is two comma-separated numbers that indicate the number of pages across and down a printed sheet. The allowable values are any two comma-separated non-zero integers. The default value is `1,1`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerBorders Property

Description

A system property used to set or get whether ToolBook pages are printed with a border around them. Setting this property is equivalent to choosing the Borders command in the Preview window.

Value

The value can be `true` or `false`. The default value is `true`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerBottomMargin Property

Description

A system property used to set or get the bottom margin of printed sheets.

Value

The value is a number that is interpreted in page units. The default value is `1440` (one inch).

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerClipText Property

Description

A system property used to set or get whether ToolBook wraps text or clips text when printing reports. ToolBook wraps text in reports at the right and bottom column or group boundaries, using the same rules it uses to wrap text in fields. ToolBook can clip text between any two characters, which allows ToolBook to print the maximum amount of text. Clipping text is useful for printing mailing labels on label stock and for printing on pre-printed forms.

Setting this property to `true` causes ToolBook to clip text.

Value

The value is `true` or `false`. The default value is `false`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerConditions Property

Description

A system property used to set or get the conditions under which fields or pages will be printed. Setting this property is equivalent to entering conditions in the Print Pages or Print Report dialog box, as described in Chapter 9, "Printing," in Using ToolBook.

You can use any valid OpenScript operators, as well as text, date, and name comparisons with the operators that include those options.

Value

The value is the expression that appears in the Pages Where text box in the Print Report or Print Pages dialog box. The expression evaluates to `true` or `false`, and determines which pages are printed or included in a report. The default value is null.

If this property is set to an invalid expression, an error occurs and ToolBook displays the Execution Suspended message box.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

Examples

```
to handle buttonUp
-- Prints all pages where the first word in field "Name"
-- is Smith

set printerConditions to \
   "word 1 of text of field ""Name""" && \
   "of this page is ""Smith"""
start spooler
   print all pages
end spooler


-- Printer condition is to print pages with a date earlier
-- than current sysDate
set printerConditions to \
   "word 1 of text of field ""Date"" < sysDate as date"
end
```

printerFieldNames Property

Description

A system property used to set or get whether field names are printed in a report. Setting this property to `true` is the equivalent of checking Print Field Names in the Print Report dialog box.

Value

The value is `true` or `false`. The default value is `true`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerFields

Property

Description

A system property used to set or get which record fields are included in a report. This property must be set before specifying values for printerFieldWidths.

Value

The value is a comma-separated list of field names. The default value is null.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerFieldWidths Property

Description

A system property used to set or get the column widths in a column report and the width of groups in a group report.

You must specify a value for the `printerFields` property before setting the `printerFieldWidths` property, or ToolBook may ignore specified widths that don’t have corresponding field names.

Value

The value is a comma-separated list of numbers which are interpreted in page units. For column reports, each number in the list represents the width of one column. For group reports, ToolBook uses the first value in the list to set the width of all groups. The default value is null, which results in balanced-width columns or groups.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerGroupsAcross Property

Description

A system property used to set or get the number of groups printed side-by-side in a group report. Setting this property is the equivalent of choosing an option in the Groups Across box in the Print Report dialog box.

Value

The value is a positive integer in the range `1` through `4`. The default value is `1`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerGutterHeight

Property

Description

A system property used to set or get the vertical distance between multiple pages printed on a single sheet, between rows in a column report, and between groups in a group report. This property has no effect when the value of the `printerArrangement` property is equal to `1,1`.

Value

The value is a number that is interpreted in page units. The default value is `360`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerGutters Property

Description

A system property used to set or get the vertical and horizontal distances between multiple pages printed on a single sheet, between rows in a column report, and between groups in a group report. This property has no effect when the value of the `printerArrangement` property is equal to `1,1`.

Value

The value is two numbers, separated by a comma. The first number represents the `printerGutterWidth` property. The second represents the `printerGutterHeight` property. Both numbers are interpreted in page units. The default value is `360,360`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerGutterWidth Property

Description

A system property used to set or get the horizontal distance between multiple pages printed on a single sheet, between rows in a column report, and between groups in a group report. This property has no effect when the value of the `printerArrangement` property is equal to `1,1`.

Value

The value is a number that is interpreted in page units. The default value is `360`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerLabelWidth Property

Description

A printer system property that specifies the width of labels in page units. The default value is 2160.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerLeftMargin Property

Description

A system property used to set or get the left margin used for printing.

Value

The value is a number that is interpreted in page units. The default value is `1440` (one inch).

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerMargins Property

Description

A system property used to set or get all four margins used for printing. This property is used by ToolBook to set or get the `printerLeftMargin`, `printerRightMargin`, `printerTopMargin`, and `printerBottomMargin` properties.

Value

The value is four numbers separated by commas that indicate the left, right, top, and bottom page margins respectively. The numbers are interpreted in page units. The default value for all margins is `1440` (one inch).

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerName Property

Description

A system property used to set or get the printer to be used for printing. Setting this property changes the default printer listed in the WIN.INI file.

It is recommended you do not change the value of this property. Because the format for printer names might vary between different versions of Windows, setting this property could cause your book to fail when it is run under a different version of Windows.

Value

The value is a comma-separated list that contains the name of a printer as it appears in the Printer Setup dialog box, the name of the printer driver, and the name of the printer port. The default value is the name of the currently selected printer in the Printer Setup dialog box, the currently selected printer driver, and the current printer port.

If the default printer is an HP LaserJet, the printer driver is HPPCL.DRV, and the printer port is LPT1:, then the value of the `printerName` property is `PCL / HP LaserJet,HPPCL,LPT1:`.

printerPageBitmap Property

Description

A printer system property that specifies the printer resolution for printing pages as a logical value, that is `false` if ToolBook should print at printer resolution, and `true` if it prints at screen resolution. The default value is `false`, which is the equivalent of unchecking Print As Bitmap in the Print Pages dialog box.

As with all printer properties, the value set for `printerPageBitmap` just before sending the first print command in a script applies for the duration of the print job.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerRightMargin Property

Description

A system property used to set or get the right margin used for printing.

Value

The value is a number that is interpreted in page units. The default value is `1440`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerSize

Property

Description

A printer system property that specifies the width and height of printer groups in a group report as a pair of positive numbers in page units. If either number is zero, the size of a printed group is unlimited in that direction. The default value is 0,0.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerStyle Property

Description

A system property used to set or get the style in which data is printed in reports. For details about report styles, see Chapter 9, "Printing," in Using ToolBook.

Value

The value must be either `pages`, `columns`, or `groups`. The default value is `pages`.

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

printerTopMargin Property

Description

A system property used to set or get the top margin used for printing.

Value

The value is a number that is interpreted in page units. The default value is `1440` (one inch).

Note:

Unlike other system properties, the value of this property is saved with a book and restored to the book's value when a book is opened.

script

Property

Description

A property used to set or get the script of an object. All objects, including books, pages, backgrounds, and hotwords, have this property.

If the `script` property is set to a value that has incorrect syntax for a script, an error occurs, ToolBook displays the Execution Suspended message box, and no changes are made to the value of the `script` property.

Scripts that get or set the value ofthe script property for any object will cause an error in Runtime ToolBook.

Value

The value is the text of the object’s script or null if the object has no script.

Example

```
set the script of button id 1 to \
   "to handle buttonUp;go to next page;end"
```

scroll

Property

Description

A field and record field property used to set or get the number of lines that are hidden above the top of a scrolling field.

Value

The value is a non-negative integer giving the number of lines that are scrolled above the topmost visible line of the field’s displayed text. The default value is `0`.

selectedText Property

Description

A system property used to set or get the text that is currently selected. If text is selected and you set the value of the `selectedText` property, ToolBook replaces the currently selected text with the value of the `selectedText` property. Setting this property when no text is selected results in an error and ToolBook displays the Execution Suspended message box.

Value

The value is a string containing the text that is currently selected. If there is no selection, the value is null.

selectedTextLines Property

Description

A field property that specifies which textlines are selected in a list box field as a list of textline numbers. The default is `null`.

Value

If the value of the `fieldType` property of a field is not `singleSelect` or `multiSelect` then the value of `selectedTextLines` is always null.

For `singleSelect` or `multiSelect` listbox fields, if no text is selected, the value of this property is null, If text is selected, the value of this property is a comma separated list of the lines selected.

Example

```
get selectedTextLines of field "Topic List"
   -- Puts a list of textline numbers into It

set selectedTextLines of field "Topic List" to 1,2,5,6
   -- Selects textLines 1, 2, 5, and 6 of a multiSelect
   -- listbox field.
```

The following example shows how to use the `selectedTextLines` property to navigate to a page when its name is clicked in a `singleSelect` listbox:

```
-- When button is clicked in this field, use the textline
-- identified by the selectedTextLines property as a page
-- name and go to it
to handle buttonUp
   if selectedTextLines <> null
      go page (textline selectedTextLines of self)
   end
end
```

selectedTextState Property

Description

A system property used to get information about the text selected in a field. 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.

Value

If no text is selected, the value of this property is null. If text is selected, the value of this property is a comma-separated list of five items:

1. Starting character number
2. Ending character number
3. Starting line number
4. Ending line number
5. true or false

The starting and ending character numbers are positive integers that indicate the ordinal position of the first and last characters of the selection, where `1` represents the first character in the field.

The starting and ending line numbers are positive integers that indicate the ordinal position of the first and last text line of the selection, where `1` represents the first text line in the field. The text lines may or may not correspond to lines on the screen, depending on how lines wrap within the field.

The last item (`true` or `false`) indicates whether the selected text contains a hotword. If the item is `true`, the selected text contains all or part of a hotword. Otherwise, the item is `false`.

Example

With the following handlers, a reader can select text in a field called `Workspace`, then click a button labeled `Underline` to underline the selected text.

```
-- This handler is in the field's script. When the
-- mouse cursor leaves the field the selectedTextState
-- property is stored in the lastTextState system
-- variable
to handle leaveField
   system lastTextState
   set lastTextState to selectedTextState
end leaveField

-- This handler is in the button's script. If there
-- is a value in the lastTextState system variable
-- the characters indicated by the first and second
-- items are underlined
to handle buttonUp
   system lastTextState
   if lastTextState <> null then
      set the fontStyle of chars (item 1 of lastTextState) \
```

```
         to (item 2 of lastTextState)\
         of text of field "workspace" to underline
   end
end buttonUp
```

selection Property

Description

A system property used to set or get the object that is currently selected. If an object is selected, the value of `selection` is the unique name of the object. If there is a multiple selection, the value of `selection` is a comma-separated list of the unique names of the selected objects. When you create a new object, the `selection` property is set to the unique name of the object.

The `selection` property can only be set to the current page or to objects on the current page. To select an object on the background, the background must have the focus (you must send the `background` message).

To unselect an object you can use the `unselect` command or the `clear` command as in `clear selection`.

For details about the unique name of an object, see the `uniqueName` property.

Value

The value is the unique name of the currently selected object or a comma-separated list of the unique names of the selected objects. If there is no selection, the value is null.

Example

```
to handle buttonUp
   draw a field from 2088,1440 to 3528,5760
   set the borderStyle of the selection to shadowed
      -- Set the border style of the new field; it is
      -- automatically selected after it is drawn so the
      -- value of the selection property will be the ID of
      -- the new field
end buttonUp
```

self

Property

Description

A system property used to get the unique name of the object whose script is currently executing.

For details about the unique name of an object, see `uniqueName`.

Value

The value is the unique name of the object whose script is currently executing. If ToolBook is executing a statement in the Command window, the value is the unique name of the currently displayed page.

singleLine Property

Description

Note:

As of ToolBook version 1.5, this property is obsolete. Instead, you should use the `singleLineWrap` value of the `fieldType` property.

A field and record field property used to set or get whether a field is a single-line field. For details about single-line fields, see Chapter 6, "Fields," in Using ToolBook.

Value

The value can be `true` or `false`. The default value is `false`.

size

Property

Description

A book property used to set or get the size of pages in a book.

Value

The value is a pair of positive numbers separated by a comma. The first number is the width of the book and the second number is the height of the book. Both numbers are interpreted in page units. The default value is determined by the values of the `startupWidth` and `startupHeight` properties.

spacing Property

Description

A field and record field property used to set or get the line spacing of text in fields.

Value

The value can be `1`, `1.5`, or `2`. The default value is the value of the `sysLineSpacing` property.

startupBook Property

Description

A system property used to set or get the name of the book that is opened when ToolBook is started, as defined in the `startupBook=` line in the [ToolBook] section of the WIN.INI file. Setting this property modifies the WIN.INI file.

Value

The default value is `null`.

startupDrawDirect

Property

Description

The startup system property that specifies the default value of `sysDrawDirect` for new books.

Value

The value can be `true` or `false`. The default value is `true`, and setting this property also modifies the `startupDrawDirect=` line in the [ToolBook] section of the user's WIN.INI file.

Example

```
set startupDrawDirect to true
```

startupHeight

Property

Description

A system property used to set or get a new book's default page height as defined in the `startupHeight=` line in the [ToolBook] section of the WIN.INI file. Setting this property modifies the WIN.INI file.

Value

The value must be a number in the range `1` through `12960`, and is interpreted in page units. The default value is `5760`.

startupSysBooks Property

Description

A system property used to set or get the default system books as defined in the `startupSysBooks=` line in the [ToolBook] section of the WIN.INI file. Setting this property modifies the WIN.INI file. These system books are loaded when an instance of ToolBook is opened.

Whenever an instance of ToolBook is opened, ToolBook reads the list of file path names in the WIN.INI file, and that list becomes the default value of the `sysBooks` property. For details about system books, see "Using System Books" in Chapter 6, "Beyond the Basics.", in Using OpenScript.

Value

The value is a comma-separated list of book names. The default value is null.

startupSysColors

Property

Description

A system startup property used to set whether ToolBook displays colors or displays only monochrome. Whenever an instance of ToolBook is opened, ToolBook reads the value from the WIN.INI file, and displays colors accordingly.

Unlike other ToolBook WIN.INI settings, `startupSysColors` is not an OpenScript property, so you cannot get or set its value with OpenScript.

Value

The value is `true` or `false`.

Example

```
[TOOLBOOK]
.
.
.
startupSysColors=false
```

startupUnits Property

Description

Note:

This property is obsolete in ToolBook versions 1.5 and later and should not be used. The default value of the `sysUnits` property is taken from the `IMeasure` setting in the WIN.INI file. If `IMeasure` is not supplied the default value of `sysUnits` is `english`. For further information on `IMeasure` see your Windows documentation.

A system property used to set or get the default units of measure (`english` or `metric`) as defined in the `startupUnits=` line in the [ToolBook] section of the WIN.INI file. Setting this property modifies the WIN.INI file.

Value

The value can be `english` or `metric` (in ToolBook version 1.0 the values were `inches` and `millimeters`).

startupWidth Property

Description

A system property used to set or get a new book's default page width as defined in the `startupWidth=` line in the [ToolBook] section of the WIN.INI file. Setting this property modifies the WIN.INI file.

Value

The value must be a number in the range `1` through `12960`, and is interpreted in page units. The default value is `8640`.

storeImage Property

Description

The background or page property that specifies whether the offscreen image of the background or page is stored automatically.

For details about this and other techniques to optimize page display speed, see Chapter 3, "Tips for ToolBook Authors," in the ToolBook Ideas booklet.

Value

The values are `true` or `false`; the default is `false`. If `storeImage` is true, a compressed image of the page or background is saved each time the author leaves the page or background; the image is not updated when a reader 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. See also `imageInvalid`.

Example

```
set the storeImage of this background to true
   -- causes the image of this background to be saved
   -- when switching from Author to Reader level,
   -- turning to another page at Author level, or saving
   -- the book at Author level..
```

storedImages Property

Description

The background or page property that contains information about the stored images available for various devices.

Value

The value is a series of textlines, each of which contains four words: `<device>`, `<size>`, `<compression ratio>`, and `<date stored>`. The value for `<device>` is the file name of the device driver as recognized in SYSTEM.INI; `<size>` is the number of bytes required to store the image, `<compression ratio>` is the ratio of size of the compressed image to the original size; and `<date>` is the date the image was stored in the format YYMMDDHHMM. This property cannot be set. The value of this property is the same as the list displayed in the Images Stored listbox in the Page Properties or Background Properties dialog box.

strokeColor Property

Description

A button, field, record field, draw object, paint object, picture, or background property used to set or get the stroke color of an object.

Value

The value is a comma-separated list of three non-negative numbers representing the hue, lightness, and saturation of a single color.

The hue value ranges from 0 to 360, corresponding to the angle in a color wheel where:

```
0 is red            60 is yellow        120 is green
180 is cyan         240 is blue         300 is magenta
```

Intermediate angles represent intermediate colors. `0` and `360` are equivalent.

The lightness value ranges from 0 to 100, with 0 representing 0% lightness (black) and 100 representing 100% lightness (white).

The saturation value ranges from 0 to 100, with 0 representing 0% saturation (gray) and 100 representing 100% (pure color).

ToolBook may have to convert the value to the closest value that represents an available color. The default value is the value of the `sysStrokeColor` property.

sysAlignment

Property

Description

A system property used to set or get the alignment of text lines in fields and record fields.

Value

The value can be `left`, `right`, `justify`, or `center`. The default value is `left`.

sysBooks

Property

Description

A system property used to set or get the books to be used as system books. ToolBook searches for handlers in the scripts of these books in the order given by the value of the `sysBooks` property, after ToolBook searches the current book's script but before it searches Windows Dynamic Link Libraries. ToolBook executes the statements in system book handlers in the context of the current page.

For details about system books, see "Using System Books" in Chapter 6, "Beyond the Basics" of **Using OpenScript**.

Value

The value is a comma-separated list of zero or more names of ToolBook files, with DOS path names if necessary. The default value is null.

sysCentered

Property

Description

A system property used to set or get the Draw Centered option.

Value

The value can be `true` or `false`. The default value is `false`.

sysChangesDB Property

Description

A system property used to set or get whether ToolBook displays the Save Changes dialog box when a reader closes a book in which changes have been made. When the `sysChangesDB` property is `true`, ToolBook displays the dialog box.

Important:

When the `sysChangesDB` property is `false`, ToolBook does not display the Save Changes dialog box, and all changes to a book are cancelled without warning when you close the book.

As for other system properties, the value of sysChangeDB persists through an instance, even when you open a new book.

A reliable way to reset `sysChangesDB` for a book is to include the statement `set sysChangesDB to true` or `restore system` in an `enterBook` handler in the book's script.

This property is useful in books which contain animation, so that the reader is not prompted to save changes which result from the animation.

Value

The value can be `true` or `false`. The default value of `true` means ToolBook displays the Save Changes dialog box when a reader or author closes a book in which changes have been made.

sysClientHandle

Property

Description

A system property used to get the window handle of the ToolBook client window, which is the portion of the main ToolBook window in which you can create and display objects. The client window does not include the title bar, menu bar, scroll bars, or window borders. This property is typically passed to a DLL so that it can calculate the location of the ToolBook client window. For details about window handles, see the Microsoft Windows Software Development Kit Programmers Reference.

Value

The value is the window handle of the client window which is assigned by Microsoft Windows.

sysCountry

Property

Description

A system property that specifies the name of the preferred country for applications. The default value is the `sCountry` value in WIN.INI. The value can contain up to 63 characters.

sysCurrency

Property

Description

A system property that specifies the currency symbol you want to use. The default value is the `sCurrency` value in WIN.INI. The value can contain up to three characters.

sysCursor Property

Description

A system property used to set or get the shape of the mouse pointer that appears on the screen. If you change the value of the `sysCursor` property, that value remains in effect until you change the value again or until the end of the ToolBook session.

The default shape results in normal ToolBook pointer behavior. That is, the cursor changes automatically to indicate hotwords in activated fields, and so forth.

Value

The value must be an expression that yields `none`, for no pointer, or an integer from `1` through `38` indicating a pointer shape.

Pointer Shapes

1 default 11 21

31

2 12

22

32

3 13

23

33

4 14

24

34

5 15

25

35

6 16

26

36

7 17

27

37

8 18

28

38

9 19

29

10 20

30

sysDate

Property

Description

A system property used to get the current system date, which is the date set in your computer.

Value

The value is the current system date in the format specified by the current value of the `sysDateFormat` property.

sysDateFormat Property

Description

A system property used to set or get the format for dates. The value defines the format of the date for the `sysDate` property, as well as the manner in which dates are formatted by the `format date` command.

Value

The value is a string of characters and numbers indicating the format for dates. The string can be any text and any combination of the formats shown in the following table. The value must be enclosed in quotation marks. The default value is determined by the settings of `sysShortDate`, `sysMorning` and `sysEvening` in the [international] section of the WIN.INI file. When the value of the `sysCountry` property is `usa`, the default value is `m/d/yy`.

Date Formats

Format	Description
`M`	The month's complete name
`MMM`	The abbreviated month's name
`m`	The month's number (1, 2, ...12)
`mm`	The month's number with a leading 0 for 1 through 9 (01, 02, ...12)
`d`	The day's number (1, 2, ...31)
`dd`	The day's number with a leading 0 for 1 through 9 (01, 02, ...31)
`y`	The year's number as an integer
`yy`	The last two digits of the year's number
`h`	The hour's number on a 12-hour clock (1, 2, ...12)
`hh`	The hour's number on a 12-hour clock with a leading 0 for 1 through 9 (01, 02, ...12)
`h24`	The hour's number on a 24-hour clock (0, 1, ...23)
`hh24`	The hour's number on a 24-hour clock with a leading 0 for 0 through 9 (00, 01, ...23)
`min`	The minute's number (00, 01, ...59)
`sec`	The second's number (00, 01, ...59)
`AMPM`	Suffix to indicate morning or afternoon. Default depends on settings in the WIN.INI file. If the value of the `sysCountry` property is `usa`, the default is AM if the time is in the morning and PM if the time is in the afternoon.
`seconds`	The number of seconds that have elapsed since 00:00:00 GMT on January 1, 1970

Examples

In the following examples, if it is 9:54 AM on March 27, 1990:

```
set sysDateFormat to "MMM dd y hh:min:sec AMPM"
put sysDate
   -- Displays MAR 27 1990 09:54:00 AM in the Command window

set sysDateFormat to "yy"
put sysDate
   -- Displays 90 in the Command window

set sysDateFormat to "mm/dd/yy"
put sysDate
   -- Displays 03/27/90 in the Command window

set sysDateFormat to "seconds"
get sysDate - (24 * 60 * 60)
   -- Subtract one day's worth of seconds
format it as "0"
   -- Make sure the seconds are not formatted with a
   -- trailing decimal point
format date it as "m/d/y"
put it
   -- Displays yesterday's date as 3/26/1990 in the
   -- Command window
```

sysDecimal

Property

Description

A system property that specifies the punctuation used to separate the fractional part of a decimal number from the whole number part. The default value is the `sDecimal` value in WIN.INI. The value can contain only one character.

The `sysDecimal` and `sysThousand` properties change the interpretation of numeric formats in ToolBook. For example, if field "Total" contains "1234.56", then `format text of field "total" as "#,##"` displays 1,234 if `sysThousand` is a comma, and 1234,56 if `sysDecimal` is a comma.

sysDrawDirect

Property

Description

The system property that specifies the default value of `drawDirect` for new objects.

Value

The value can be `true` or `false`. The default is the value of `startupDrawDirect`; `if no value for startupDrawDirect` is defined in the [ToolBook] section of the author's WIN.INI file, the default is `true`.

Example

```
set sysDrawDirect to false
   -- New objects will have their drawDirect property
   -- set to false
```

sysError Property

Description

A system property used to set or get the nature of a non-fatal error. A non-fatal error is an error that, although it may have consequences in script execution, does not cause ToolBook to display the Execution Suspended message box.

Several commands set the `sysError` property when they detect an unusual or exceptional condition. For example, the `createFile` command sets this property to `read only` when an attempt is made to create a file that exists and has the read-only attribute. ToolBook also sets the `sysError` property when the `sysSuspend` property is set to `false` and an error occurs that would normally cause ToolBook to display the Execution Suspended message box. When the value of sysError is set by ToolBook, a corresponding value for `sysErrorNumber` is also set.

You can set this property from within a handler.

Value

The value is a string that describes the cause of the last error. The default value is null.

sysErrorNumber Property

Description

A system property that specifies what kind of error has occurred as an integer that represents a corresponding error message, providing a language-independent method of ascertaining the exact error that has occurred.

value

The default value of `sysErrorNumber` is 0, which corresponds to a null value for `sysError`. `Whenever ToolBook sets sysError` to `null` (such as after a successful search), `sysErrorNumber` is set to 0. As with `sysError`, `sysErrorNumber` may be set, but must be an integer.

The `sysErrorNumber` property helps authors writing internationalized books find out what error has occurred without having to know the local language. An author can use this property is the same manner as `sysError`, where `sysSuspend` is set to `false`, `sysError` is set to `null`, an action is taken, then `sysSuspend` is set to `true` and `sysError` is tested. If `sysError` is not `null`, an error has occurred. When you create similar scripts using `sysErrorNumber`, test to find whether its value is not 0.

Note:

The `sysErrorNumber` property is also set to the HIWORD of the long return value for a DLL function linked as returning a STRING type when the LOWORD is zero. This is identical to the setting of `sysError` in this circumstance.

The following table lists the possible numeric values for the `sysErrorNumber` property and the text that `sysError` will assigned. Many of the text values shown are templates for the actual values, such as a specific filename or object name.

Values for sysErrorNumber and related error messages

2	The requested page is not in this book.
3	The requested file is not loadable by this application.
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	Not enough file memory. Close other applications or save this book and try again.
13	Not enough memory. Close other applications or save this book and try again.
15	Not enough local memory. Save this book and open it again.
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.
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.
30	This object is in use and cannot be removed.
33	Call to Windows (GDI) failed. Memory is probably low. Close this book and try again.

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?
49	A printer driver is not installed. Run the Control Panel to install a printer.
56	Number is outside the valid range for this command.
57	Call to Windows function (MakeProcInstance) failed.
60	Error while reading from the Clipboard. Cannot continue operation.
61	Error while writing to the Clipboard. Cannot continue operation.
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.
69	The maximum number of hotwords allowed in a field is 255.
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.
77	A file with that name is currently in use. Try a different name.
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	Cannot find specified search text.
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.
92	This page contains no record fields.
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	No pages match the specified conditions.
105	Cannot merge a book into itself.
106	Groups cannot be nested more than 16 levels deep.
107	Not a valid number format.
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.
558	File already open.
559	No such file.
560	Read only.
565	End of file.
627	File exists.
641	Not found.
642	OK.
643	File not open.
4003	That password is not correct.
4004	Name contains an invalid character.
8011	Cannot find page number %ld.
8012	Cannot find page "%.100s".

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.
8032 There is no object named %.100s.
8033 No such property for object %.100s.
8034 Cannot set object property as requested.
8035 Cannot get object property as requested.
8040 Not a graphic object: %.100s.
8042 Not a rectangle: %.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.
8057 Not a valid file name: %.100s.
8058 Not enough memory. Try quitting and restarting ToolBook.
8059 File already open.
8060 No such file.
8061 Read only.
8065 File read/write error.
8066 End of file.
8068 Error when starting spooler.
8071 Magnification power must be 1, 2, 4, 8 or 16.
8077 Cannot get argument %d. Handler has only %d arguments.
8079 Not a container: %.100s.
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
8093 Unable to format date.
8094 Unable to format time.
8095 Value "%.100s" is out of range.
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.
8103 Cannot have more than 60 menus and 255 items per menu.
8104 Menu already exists: "%.100s".

8105 Menu item already exists: "%.100s".
8106 No menu named "%.100s".
8108 Cannot load DLL "%.100s".
8110 Error: %.100s.
8111 Encountered bad sort key on "%.100s".
8112 No search to repeat.
8116 Too many objects selected for the requested operation.
8118 Calls to handlers are nested too deeply. Try making fewer nested calls.
8120 Print command must be within a Start Spooler control structure.
8123 OK
8124 Failed: Denied
8125 Failed: Busy
8126 Failed: Memory Error
8127 Failed: No Server
8128 Failed: Interrupted
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.
8141 Not a positive integer: %.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 location: %.100s.
8147 Not a book, page or background: %.100s.
8148 Not a window or movable object: %.100s.
8149 Not a page or background: %.100s.
8150 Not a book: %.100s.
8151 Not an object: %.100s.
8152 The object with this ID is not a(n) %.100s.
8155 Cannot find system book "%.100s".
8156 Cannot create record field.
8159 No record fields found to sort.
8160 Cannot change pages in this book.
8161 Record too large.
8162 Not a valid object reference.
8164 Cannot select text of "%.100s". It is not a field or record field.
8166 Cannot nest Start Spooler commands.
8168 DLL "%.100s" is not loaded.
8169 No function named "%.50s" in DLL "%.100s".
8170 No function at ordinal %d in DLL "%.100s".
8171 Wrong number of arguments for DLL function.
8172 Wrong type of argument for DLL function.
8173 Cannot get printer property.
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.
8180 Undo is not available in this context.

8181 Group is not selected. Cannot ungroup.
8182 Cannot perform desired Clipboard operation.
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 This string is too long: "%.100s".
8193 Error handled.
8194 Menu identifier %.100s is not a valid menu or menu item name.
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 record field on the current page: %100.s
8202 The valid range for sysCursor is 1 - 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.
8214 Specified file is not an icon.

sysEvening

Property

Description

A system property that specifies the string following time after noon in a 12-hour format.

Value

The default value is the s2359 value in WIN.INI. The value can contain up to two characters. The value of this property affects the way values are displayed when they are formatted as dates.

sysFillColor Property

Description

A system property used to set or get the system default fill color.

Value

The value is a comma-separated list of three non-negative numbers representing the hue, lightness, and saturation of a single color.

The hue value ranges from 0 to 360, corresponding to the angle in a color wheel where:

```
0  is red            60  is yellow        120  is green
180  is cyan         240  is blue          300  is magenta
```

Intermediate angles represent intermediate colors. 0 and 360 are equivalent.

The lightness value ranges from 0 to 100, with 0 representing 0% lightness (black) and 100 representing 100% lightness (white).

The saturation value ranges from 0 to 100, with 0 representing 0% saturation (gray) and 100 representing 100% (pure color).

ToolBook may have to convert the value to the closest value that represents an available color. The default value is `0,100,0` (white).

sysFontFace Property

Description

A system property used to set or get the default typeface for text typed in fields.

If this property is set to a value that specifies a typeface not available in the Character dialog box, the font typeface in effect before the `sysFontFace` property is set remains the object's default typeface. If, at a later time, the typeface specified as the value for the property becomes available, that typeface is used.

Value

The value is the name of the typeface for the button label or the field text as shown in the Character dialog box. The default value is `system`.

sysFontSize

Property

Description

A system property used to set or get the default type size for text typed in fields. The number value is interpreted as the point size for text.

If this property is set and the value specifies a point size that is not available for the system typing font, the next smaller value is used.

Value

The value must be a positive integer that defines the point size of the current system typing font as shown in the Character dialog box. The default value is `9`.

sysFontStyle Property

Description

A system property used to set or get the default type style for text typed in fields.

Value

The value must be null or a comma-separated list of one or more of the values `bold`, `italic`, `underline`, and `strikeout`. The combination of these values describes the style of the object's default font. A null value is equivalent to the plain style. The default value is null.

sysGrid

Property

Description

A system property used to set or get whether the grid is visible.

Value

The value can be `true` or `false`. The default value is `false`.

sysGridSnap Property

Description

A system property used to set or get whether objects snap to the grid lines when objects are created, moved, or sized.

Value

The value can be `true` or `false`. The default value is `false`.

sysGridSpacing Property

Description

A system property that indicates the horizontal and vertical distance between grid lines.

Value

The value must be a positive integer in the range `30` through `4320`. The number is interpreted in page units. The default value is `180`.

sysHistory Property

Description

A system property used to set or get the items in the history stack, which is a comma-separated list of the unique names of up to the last 100 pages visited during the current ToolBook instance. As ToolBook displays each page, it pushes the unique name of the page onto the value of this property. Only expressions that evaluate to the unique name of a page can be pushed onto the value of the `sysHistory` property.

For details about pushing and popping items, see the `push` and `pop` commands. Also see the `sysHistoryRecord` property.

Value

The value is a comma-separated list of the unique names of the last 100 pages visited in the current ToolBook session. ToolBook uses the value of this property to determine the content of the History dialog box.

sysHistoryRecord Property

Description

A system property used to determine whether ToolBook adds pages to the history stack of the `sysHistory` property. The default value `true` allows pages to be added to the history.

The history of an instance is a comma-separated list of up to the last 100 unique pages in the order of how recently they were displayed. While `sysHistoryRecord` is set to `false`, any pages viewed or accessed are not added to the history of the instance, but the History dialog box can still be displayed.

One way this property can be used is to create a custom history that ToolBook can't modify. This property can also be useful for hiding information from a reader. For example, you can set `sysHistoryRecord` to `false` to keep a reader from finding out which pages are accessed by a script. Setting `sysHistoryRecord` to `false` also speeds up animated sequences that involve flipping pages.

A side effect of setting `sysHistoryRecord` to `false` is that, because the Back command on the Page menu always goes to the last page added to the history, a reader may find that choosing Back does not always go back to the previous page accessed. If you set `sysHistoryRecord` to `false`, it might be a good idea to deactivate the Back command.

Value

The value can be `true` or `false`. The default value is `true` which allows pages to be added to the history.

sysHotwordsShown Property

Description

A system property used to get or set whether hotwords in fields are outlined so that a reader can identify them.

Value

The value can be `true` or `false`. The default value is `false`.

sysICountry

Property

Description

A system property that specifies the country code.

Value

This is the same as the country's international telephone code, except for Canada, which is 2. The default value is the `iCountry` value in WIN.INI.

sysICurrDigits

Property

Description

A system property that specifies the preferred number of digits after the decimal separator in currency.

Value

The default value is the `iCurrDigits` value in WIN.INI.

sysICurrency

Property

Description

A system property that specifies the preferred format for the currency symbol.

Value

The value can be an integer from 0 through 3, where:

0 = $2 1 = 2$ 2 = $ 2 3 = 2 $

The default value is the `iCurrency` value in WIN.INI. The currency symbol is specified by the value of `sysCurrency`.

sysIDate

Property

Description

A system property that specifies the order of items formatted as dates.

Value

The value can be an integer from 0 through 2:

0 specifies the order month, day, year (as in 12/31/90)

1 specifies the order day, month, year (as in 31/12/90)

2 specifies the order year, month, day (as in 90/12/31)

The default value is the `iDate` value in WIN.INI. The character used to divide the items in dates is specified by the value of `sysShortDate`. This setting is not used by Windows 3.0 and exists only for compatibility with Windows 2.x.

sysIDigits Property

Description

A system property that specifies the number of digits displayed after the decimal separator.

Value

The default value is the `iDigits` value in WIN.INI.

sysILZero Property

Description

A system property that specifies whether decimal numbers should include a leading zero.

Value

The value can be 0 or 1:

0 Omit leading zero (such as .7)

1 Include leading zero (such as 0.7)

The default value is the `iLZero` value in WIN.INI. The actual decimal separator is specified by the `sysDecimal` setting.

sysIMeasure Property

Description

A system property that specifies the preferred measurement system, where 0 is metric (centimeters) and 1 is english (inches).

Value

The default value is the `iMeasure` value in WIN.INI.

sysIndents

Property

Description

A system property used to set or get the paragraph indents for fields.

Value

The value is three non-negative integers separated by commas. The first number is the first line indent, the second is the left indent for a paragraph (or textline), and the last is the right indent for a paragraph. The numbers are interpreted in page units. The default value is `0,0,0`.

sysINegCurr

Property

Description

A system property that specifies the preferred negative number format for the currency symbol.

Value

The value can be an integer from 0 through 7, representing these formats:

0 = ($1)	2 = $-1	4 = (1$)	6 = 1-$
1 = -$1	3 = $1-	5 = -1$	7 = 1$-

The default value is the `iNegCurr` value in WIN.INI. The currency symbol is specified by the value of `sysCurrency`.

sysITime

Property

Description

A system property that specifies whether a 12-hour or 24-hour clock is preferred. Setting this property in ToolBook has no effect. It simply reports what was in the WIN.INI file for the `iTime` setting.

Value

An integer:

0 12-hour clock (as in 1:00)

1 24-hour clock (as in 13:00).

The default value is the `iTime` value in WIN.INI.

sysITLZero

Property

Description

A system property that specifies whether to put leading zeros in time.

Value

An integer:

0 Omit a leading zero in the time (e.g. 9:15)

1 Put a leading zero in the time (e.g. 09:15)

The default value is the `iTLZero` value in WIN.INI.

sysLanguage Property

Description

A system property that specifies the language used by Windows.

Value

The default value is the `sLanguage` value in WIN.INI, which is used by Windows applications for language-specific tasks, such as sorting or spell-checking. The following values are allowed:

dan = Danish	ice = Icelandic
dut = Dutch	itn = Italian
eng = International English	nor = Norwegian
fcf = French Canadian	por = Portuguese
fin = Finnish	spa = Spanish
frn = French	swe = Swedish
ger = German	usa = U.S. English

sysLevel

Property

Description

A system property used to set or get the working level.

Value

The value can be `author` or `reader`. The default value is the current working level.

Scripts that set sysLevel to author will cause an error in Runtime ToolBook.

sysLineSpacing Property

Description

A system property used to set or get the line spacing of newly typed text in fields.

Value

The value can be `1`, `1.5`, or `2`. The default value is `1`.

sysLineStyle Property

Description

A system property used to set or get the style of borders of newly created rectangles, ellipses, lines, curves, arcs, angled lines, polygons, irregular polygons, paint objects, pictures, pies, and rounded rectangles.

Value

The value can be `none`, for no visible line; `0`, `1`, `2`, `3`, `4`, `6`, or `8`, representing the width of the line in pixels; `dashed` for a dashed line style; or `dotted` for a dotted line style. The default value is `1`.

sysList

Property

Description

A system property that specifies the character used to separate record items for data files imported into ToolBook. This property does not affect lists in OpenScript statements.

Value

The default value is the `sList` value in WIN.INI. The value can contain only one character.

sysLockScreen Property

Description

A system property used to set or get whether to allow screen updates during the execution of the current handler. The default value of `false` allows normal screen updates.

By setting `sysLockScreen` to `true` in a script, you can have the script make several changes to the screen and then show all the changes at once, instead having ToolBook update the screen after each change. For example, to have an application neatly set up the screen when a reader opens a book, you can write statements in the book's `enterBook` handler to set `sysLockScreen` to `true`, prepare the screen, then set `sysLockScreen` to `false`.

You can also use this property to force ToolBook to update the screen. To update the screen, set `sysLockScreen` first to `true`, then to `false`.

If you set `sysLockScreen` to `true` in a handler, ToolBook automatically resets `sysLockScreen` to `false` when it exits the top-most calling handler.

The `sysLockScreen` property does not affect the Command window or the palettes.

Value

The value can be `true` or `false`. The default value is `false`.

Example

```
to handle buttonUp
   -- This script is for a button that adds a shrub to a landscape
   set sysLockScreen to true
      -- Freeze the screen
   set sysSuspendMessages to true
      -- Prevent ToolBook from sending leave and enter messages
   go to page 1 of book "images.tbk"
      -- Go to a book of clip art
   select group "shrub"
      -- Select the desired image
   send copy
      -- Copy the image
   send back
      -- Return to the original page
   set sysSuspendMessages to false
      -- Restore normal message behavior
   set sysLockScreen to false
      -- Unfreeze the screen
   send paste
      -- Paste the desired image
end buttonUp
```

sysLongDate

Property

Description

A system property that specifies the preferred long date format.

Value

The value can contain up to 31 characters, including the following character combinations which specify the formats indicated here:

M = 1-12	dd = 01-31
MM = 01-12	ddd = Abbreviated weekday name
MMM = Abbreviated month name	dddd = Weekday name
MMMM = Month name	yy = 00-99
d = 1-31	yyyy = 1900-2040

The default value is the `sLongDate` value in WIN.INI.

sysMagnification Property

Description

A system property used to get whether the view of the main ToolBook window has been zoomed. To change the magnification, use the `magnify` command.

Value

The value can be `1`, `2`, `4`, `8`, or `16`, indicating the power of magnification of the main window.

Example

```
if sysMagnification <>1
   -- Check whether view is zoomed
   magnify 1
      -- Restore view to normal magnification
end if
```

sysMorning

Property

Description

A system property that specifies the string following times before noon in a 12-hour time format.

Value

The default value is the `s1159` value in WIN.INI. The value can contain at most two characters. The value of this property affects the way values are displayed when they are formatted as dates.

sysMousePosition Property

Description

A system property used to set or get the position of the pointer.

Value

The value is a pair of comma-separated integers that gives the coordinates of the location of the pointer on the page. The first number is the horizontal coordinate, and the second number is the vertical coordinate, measured from the top left corner of the page. The numbers are interpreted in page units.

sysNumberFormat Property

Description

A system property that describes the manner in which numbers are formatted by the `format number` command.

When ToolBook tries to read a formatted number, it strips out dollar signs ($) and percent signs (%) in the number and ignores spaces. If any other characters are included in the formatted number, ToolBook does not recognize the string as a number.

Value

You can define the number format you want by building a string containing any of the following symbols. The string must be enclosed in quotes. The default value is based on local values for `sysThousand`, `sysDecimal`, `sysDigits`, and `sysILZero`. If the value of the `sysCountry` property is `usa`, the default is `null`.

Number format characters

Symbol	Meaning
`null`	Default format. ToolBook displays the number as precisely as possible, using a period as the decimal separator, and using scientific notation when necessary.
?	General precision format. If used as `<new format>`, ToolBook displays the number as precisely as possible using the value of `sysDecimal` as the decimal separator. If used as `<old format>`, ToolBook will discard characters matching `sysCurrency` and `sysThousand`, and replace the `sysDecimal` character with a period before interpreting the number.
`0`	Digit placeholder. If the number has fewer digits on either side of the decimal point than there are zeros on either side of the decimal point, ToolBook displays the extra zeros. If the number has more digits to the right of the decimal point than there are zeros to the right of the decimal point in the format, ToolBook rounds the number to as many decimal places as there are zeros to the right of the decimal point in the format. If the number has more digits to the left of the decimal point than there are zeros to the left of the decimal point in the format, ToolBook displays the extra digits.
#	Digit placeholder. Similar to 0 above, except that ToolBook does not display extra zeros if the number has fewer digits on either side of the decimal point than there are #s on either side of the decimal point in the format.
`. or ,`	Decimal separator depending on the value of the `sysDecimal` property. If the format contains only #s to the left of this symbol, ToolBook begins numbers smaller than 1 with the decimal separator. To avoid this, use 0 as the first digit placeholder to the left of a decimal separator.

, or .	Thousands separator depending on the value of the sysThousand property. ToolBook separates thousands by the thousands separator if the format contains the separator surrounded by #s or 0s.
E- E+ e- e+	Scientific notation. If a format contains one 0 or # to the right of an E-, E+, e-, or e+, ToolBook displays the number in scientific notation and inserts an E or e. The number of 0s or #s to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign by negative exponents. Use E+ or e+ to place a minus sign by negative exponents and a plus sign by positive exponents.
<character>	Literal text. Any printable character besides the above symbols is displayed literally. For example, if you want a dollar sign to precede all numbers, you can specify a format like "$###.00". These characters cannot be imbedded in the number format; literal text must be before or after the group of format characters.

Example (sysCountry is set to USA)

Format	Number typed and resulting display		
	2	-2	.2
""	2	-2	0.2
"0"	2	-2	0
"0.00"	2.00	-2.00	0.20
"#,##0"	2	-2	0
"#,##0.00"	2.00	-2.00	0.20
"0.00E+00"	2.00E+00	-2.00E+00	2.00E-01
"$##.00"	$ 2.00	$- 2.00	$ .20
"It is 0"	It is 2	It is -2	It is 0

Use decimal tabs or a monospace font to ensure alignment of numbers in columns. Proportionally spaced fonts, such as the one used in the preceding table, can result in uneven space padding for numeric formats.

```
set sysNumberFormat to "000.##"
   -- Means that 34.6639 would be formatted as
   -- 034.66 if formatted as sysNumberFormat
```

sysOperatingSystem Property

Description

A system property that specifies the current operating system, graphical environment, and version number. This property is gettable but not settable.

Value

"OS/2 PM 1.20"	Presentation Manager
"DOS Windows 3.0"	Windows

sysPageScroll Property

Description

A system property used to set or get the offset of the main window from the upper left corner of a page.

Value

The value is a comma-separated pair of numbers that gives the coordinates of the upper left corner of the window on the page. The first number is the horizontal coordinate and the second number is the vertical coordinate of that point, measured from the top left corner of the page. The numbers are interpreted in page units. The coordinates must be positive and the value must be set so that the window only displays the area inside a page.

Example

```
to handle enterPage
   set sysPageScroll to 1440,1440
      -- Results in top and left 1" of page being
      -- outside of the display area of the window
end enterPage
```

sysPasswords Property

Description

A system property used to set or get a list of encrypted passwords for ToolBook to check before requesting a password from the user. If ToolBook is looking for an Author, Open, or Save password and the encrypted form of the password appears in the value of the `sysPasswords` property, ToolBook continues without asking for a password. Otherwise, ToolBook displays the Ask Password dialog box.

By setting this property, you can 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.

The encypted form of a string is placed in the special variable, `It`, as a result of the the `ask password` command.

Value

The value is up to ten encrypted passwords, each on a separate text line. The default value is null.

Example

This example shows how an author can establish a set of passwords for several books when a reader correctly enters a password. This saves the reader from having to repeat the passwords for each book.

```
to handle enterBook
   if "retlaw" is not in sysPasswords
      -- "retlaw" is an encrypted password
      ask password "Enter your password."
      if it is "nitram"
         -- "nitram" is another encrypted password
         set sysPasswords to passwordList of this book
            -- passwordList is a user-defined property
            -- of the book, created by the author, which
            -- contains a set of encrypted passwords,
            -- each on its own line
      else
         request "Wrong password. Exiting ToolBook."
         send exit
      end if
   end if
end enterBook
```

sysPattern

Property

Description

A system property used to set or get the default pattern when graphic objects are created.

Value

The value must be `none`, `solidStroke`, `solidFill`, or an integer in the range from `1` through `128` or `253` through `255`. For the values `253` through `255`, `253` = `none`, `254` = `solidFill`, and `255` = `solidStroke`. The default value is `solidFill`.

sysPolygonShape Property

Description

A system property used to set or get the number of sides created when a regular polygon is drawn.

Value

The value must be a non-negative integer equal to the number of sides for a regular polygon. The value can be in the range from `3` through `99`. The default value is `4`.

sysRuler Property

Description

A system property used to set or get whether the rulers are displayed.

Scripts that set this property will cause an error in Runtime ToolBook.

Value

The value can be `true` or `false`. The default value is `false`.

sysRuntime

Property

Description

A system property used to get whether a reader is using the runtime version of ToolBook.

Value

The value can be `true` or `false`. The value is `true` if the reader is using the runtime version of ToolBook.

sysShortDate Property

Description

A system property that specifies the preferred short date format.

Value

The value can contain up to 15 characters and can include the special character combinations described under the entry for `sysLongDate`. The default value of sysShortDate is the sShortDate value in WIN.INI. The `sysShortDate` property sets the value for `sysDateFormat`.

sysStrokeColor Property

Description

A system property used to set or get the system default stroke color.

Value

The value is a comma-separated list of three non-negative numbers representing the hue, lightness, and saturation of a single color.

The hue value ranges from 0 to 360, corresponding to the angle in a color wheel where:

```
0  is red            60  is yellow        120  is green
180  is cyan         240  is blue          300  is magenta
```

Intermediate angles represent intermediate colors. 0 and 360 are equivalent.

The lightness value ranges from 0 to 100, with 0 representing 0% lightness (black) and 100 representing 100% lightness (white).

The saturation value ranges from 0 to 100, with 0 representing 0% saturation (gray) and 100 representing 100% (pure color).

ToolBook may have to convert the value to the closest value that represents an available color. The default value is `0,0,0` (black).

sysSuspend Property

Description

A system property used to set or get whether ToolBook displays the Execution Suspended message box when an error occurs using a script. When the `sysSuspend` property is `true`, ToolBook suspends script execution in the event of an error. When the `sysSuspend` property 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 handlers to check the value of `sysError` and respond appropriately.

Note:

When the `sysSuspend` property is set to `false`, script execution will continue even if an error occurs. This could cause unpredictable behavior in a book.

Value

The value can be `true` or `false`. The default value of `true` means ToolBook displays the Execution Suspended message box when an error occurs during script execution.

sysSuspendMessages Property

Description

A system property used to set or get whether ToolBook sends messages automatically. When this property is `false`, ToolBook sends messages normally.

When this property is `true`, ToolBook does not send system-generated messages. System-generated messages include all built-in event messages, such as `leavePage` and `make`, and all menu messages.

By setting this property to `true` within a handler, you can prevent ToolBook from sending event messages such as `leavePage`, `enterBook`, `leaveBook`, and `enterPage` when the handler (or other handlers it calls) navigates to other pages or books.

ToolBook resets the `sysSuspendMessages` property to `false` when the top-most calling handler finishes executing.

Value

The value can be `true` or `false`. The default value is `false`.

Example

```
to handle buttonUp
   -- This script is for a button that adds a shrub to a landscape
   set sysLockScreen to true
      -- Freeze the screen
   set sysSuspendMessages to true
      -- Prevent ToolBook from sending leave and enter messages
   go to page 1 of book "images.tbk"
      -- Go to a book of clip art
   select group "shrub"
      -- Select the desired image
   send copy
      -- Copy the image
   send back
      -- Return to the original page
   set sysSuspendMessages to false
      -- Restore normal message behavior
   set sysLockScreen to false
      -- Unfreeze the screen
   send paste
      -- Paste the desired image
end buttonUp
```

sysTabSpacing Property

Description

A system property used to set or get the default tab stop settings for fields.

Value

The value is a positive number that gives the interval for tab stops in fields. The number is interpreted in page units. The default value is `720`.

sysTabType

Property

Description

A system property used to set or get the default tab type for fields. Tabs can be left-aligned or decimal-aligned.

Value

The value is either `left` or `decimal`. The default value is `left`.

sysThousand Property

Description

A system property that specifies the symbol used to separate thousands in a number with more than three digits.

Value

A single character (For example, if the value is "," (comma), 3000 is displayed as 3,000.) The default value is the `sThousand` value in WIN.INI. A blank can be specified as the separator.

The sysThousand and sysDecimal properties change the interpretation of numeric formats in ToolBook. For example, if field "Total" contains "1234.56", then `format text of field "total" as "#,##"` displays 1,234 if `sysThousand` is a comma, and 1234,56 if `sysDecimal` is a comma.

sysTime

Property

Description

A system property used to get the current system time. The system time is the same as the DOS system time.

Value

The value is the current system time in the format specified by the value of the `sysTimeFormat` property.

sysTimeChar Property

Description

A system property that specifies the preferred character used to separate the hours, minutes, and seconds in time.

Value

A single character. If the value is a ":" (colon), 15:29:31 is displayed. The default value is the `sTime` value in WIN.INI.

sysTimeFormat Property

Description

A system property used to set or get the format for times. The value defines the format of the value of the `sysTime` property, as well as the manner in which the time is formatted by the `format time` command.

Value

The value is a string of characters and numbers indicating the format for time. The string can be any combination of the formats shown in the following table. The default format is determined by the settings of `sysITime` and `sysTimeChar` in the international section of the WIN.INI file. If the value of the `sysCountry` property is set to `usa`, the default format is `h:min:sec`.

Time formats

Format	Description
`h`	The hour's number on a 12-hour clock (1, 2, ...12)
`hh`	The hour's number on a 12-hour clock with a leading 0 if it is 1 through 9 (01, 02, ...12)
`h24`	The hour's number on a 24-hour clock (0, 1, ...23)
`hh24`	The hour's number on a 24-hour clock with a leading 0 if it is 0 through 9 (00, 01, ...23)
`min`	The minute's number (00, 01, ...59)
`sec`	The second's number (00, 01, ...59)
`AMPM`	Suffix to indicate morning or afternoon. Default depends on settings in the WIN.INI file. If the value of the `sysCountry` property is set to `usa`, the default is AM if the time is in the morning and PM if the time is in the afternoon.
`seconds`	The number of seconds that have elapsed since 00:00:00 GMT on January 1, 1970

Examples

In the following examples, if the time is 9:54 in the evening:

```
set sysTimeFormat to "h:min:sec AMPM"
put sysTime
   -- Displays 9:54:00 PM in the Command window

set sysTimeFormat to "min"
put sysTime
   -- Displays 54 in the Command window
```

sysTransparent Property

Description

A system property used to set or get the default for the `transparent` property of new objects.

Value

The value must be `true` or `false`. The default value is `false`.

sysUnits Property

Description

A system property used to set or get the units of measure used in the page rulers.

Value

The value must be `english` or `metric`. The default value is based on the the value of the `iMeasure` setting in the [International] section of WIN.INI.

Note:

The values `inches` and `millimeters` are obsolete in ToolBook versions 1.5 and later and should not be used.

sysVersion

Property

Description

A system property that specifies the currently running version of ToolBook. This property is gettable but not settable.

sysWindowHandle

Property

Description

A system property used to get the window handle of the ToolBook main window. For details about window handles, see the Microsoft Windows Software Development Kit Programmers Reference.

Value

The value is the window handle of the main ToolBook window as assigned by Microsoft Windows.

tabSpacing Property

Description

A field and record field property used to set or get the tab stop settings in a field or record field. Setting the value of this property is the equivalent of entering a value in the Interval box in the Paragraph dialog box when a field or record field is selected.

Value

The value is a positive integer that gives the interval for tab stops in a field. The number is interpreted in page units. The default value is the value of the `sysTabSpacing` property.

tabType

Property

Description

A field and record field property used to set or get whether left or decimal tabs are used in a field or record field. Setting the value of this property is the equivalent of choosing Left Tabs or Decimal Tabs in the Paragraph dialog box when a field or record field is selected.

Value

The value is either `left` or `decimal`. The default value is the value of the `sysTabType` property.

target Property

Description

A system property used to get the object that received the current message.

Value

The value is the unique name of the object that received the current message.

text Property

Description

A field or record field property used to set, get, or manipulate the text in a field or record field. Also a hotword property used to get the text of a hotword.

For details about selecting text, see the `select` command.

Value

The value is a string consisting of the text in a specified field, record field, or hotword.

Examples

```
set text of field "Response" to "That's correct."
   -- Puts "That's correct." in the Response field on
   -- the current page

clear textLine 1 of text of field "Names" of page "Contacts"
   -- Deletes the first line of text from the Names
   -- field on the Contacts page
```

textAlignment

Property

Description

A field and record field property used to set or get the alignment of text in a field or record field.

Value

The value can be `left`, `right`, `justify`, or `center`. The default value is `left`.

textOverflow Property

Description

A button, field, and record field property used to get the number of characters clipped by the bottom boundary of a field or the bottom or right boundry of a button, including characters only partially visible. This property is useful if you need to find out if all of the text of a field or button fits in its display area.

Value

The value is an integer that equals the number of characters of a field's or button's text that are partially or completely clipped. The default value is `null`.

transparent

Property

Description

A button, field, or graphic property used to set or get whether an object is transparent. Unlike invisible and hidden objects, a transparent object can receive keyboard and mouse messages.

Value

The value can be `true` or `false`. The default value is `false`.

uniqueName Property

Description

An object property used to get the unique name of an object. For details about the unique names of objects, see "Referring to Objects" in Chapter 2, "Script Basics" of Using OpenScript.

Value

The value is a string containing the unique name for the object. For example, if a button is the first object created on the first page of book CONTACTS.TBK, then the `uniqueName` of the button is `button id 0 of page id 0 of book "C:\CONTACTS.TBK"`.

userProperties

Property

Description

A button, field, group, hotword, page, background, book, and graphic object property used to get the names of an object's user-defined properties that don't have corresponding `to set` handlers. For details about the use of this property, see "Defining Your Own Properties" in Chapter 6, "Beyond the Basics" of Using OpenScript.

Value

The value is a comma-separated list of the names of an object's user-defined properties that don't have corresponding `to set` handlers. The default value is null.

Example

```
put the userProperties of this page into the commandWindow
   -- Displays a list of the user-defined properties for
   -- the current page
```

vertices Property

Description

A button, field, group, window, and graphic object property used to set or get the locations of the vertices of an object. This property can also be used to get the locations of the vertices of the Command window and the palettes.

Value

For buttons, fields, groups, windows, and graphic objects other than angled lines, curves, and irregular polygons, the value is four comma-separated numbers representing the coordinates for the upper left and lower right corners of the object. For angled lines and irregular polygons, the vertices are comma-separated numbers that give the locations of endpoints of the.line segments in the object. For curves, the vertices are a comma-separated list of the locations of the reshape handles. All values for objects are expressed in page units. All values for windows and palettes are expressed as screen coordinates.

This property can be used to reshape graphic objects. For example, changing the vertices of an irregular polygon will reshape the polygon.

Example

```
set items 3 to 4 of the vertices of irregularpolygon id 6 \
   to 2463, 3546
   -- Sets the second angle of the polygon
```

visible Property

Description

A button, field, graphic, group, palette, and Command window property used to indicate whether a window or an object is shown on the screen. An object whose `visible` property is set to `false` does not receive keyboard or mouse messages, nor can it be selected or included in a search.

Also see the `hide` and `show` commands in this chapter.

Scripts that set the `visible` property to true for palettes or the Command window will cause an error in Runtime ToolBook.

Value

The value can be `true` or `false`. The default value is `true`.

Special Terms

Quick reference

Alphabetical List

Working Levels

author
both
reader

Special Variables

argCount
argList
It

Style values for object properties

black
bold
checkbox
columns
dashed
decimal
dotted
groups
italic
multiSelect
none
noWrap
pushButton
radioButton
rectangle
rounded
scrolling
shadowed
singleLineWrap
singleSelect
solidFill
solidStroke
strikeout
underline
white
wordWrap

Object type names and tool names

angledLine
arc
background
book
button
curve
ellipse
field
group
hotword
irregularPolygon
line
page
paintObject
picture
pie
polygon
recordField
rectangle
roundedRectangle

Window and palette names

colorTray
commandWindow
linePalette
mainWindow
menuBar
patternPalette
polygonPalette
scrollBar
spectrumPalette
statusBox
toolPalette

Ordinal numbers

first
second
third
fourth
fifth
sixth
seventh
eighth
ninth
tenth

Relational terms

first
last
mid
middle
next
previous

Units of measure

english
metric

Alignment and direction

ascending
bottom
center
descending
horizontal
justify
left
right
top
vertical

Text and string specifiers

char[s]	item[s]	textLine[s]
character[s]	text	word[s]

Articles and prepositions

a	by	onto
after	for	the
an	from	this
at	into	to
before	of	with

Logical Values

false	true

Miscellaneous

again	id	page
alias	locateOnly	pages
all	my	system
date	name	time
end	none	
excluding	number	

Special Terms - Alphabetical list

Quick reference

a
after
again
alias
all
an
angledLine
arc
ascending
at
author
background
before
black
bold
book
both
bottom
button
by
center
char
character
characters
chars
checkbox
colorTray
columns
commandWindow
curve
dashed
date
decimal
descending
dotted
eighth
ellipse
end
english
excluding
false
field
fifth
first
for
fourth
from
gray
group
groups
horizontal
hotword
id
into
irregularPolygon
italic
item
items
justify
last
left
line
linePalette
locateOnly
mainWindow
menuBar
metric
mid
middle
multiSelect
my
name
next
ninth
none
noWrap
number
of
onto
page
pages
paintObject
patternPalette
picture
pie
polygon
polygonPalette
previous
pushbutton
radiobutton
reader
recordField
rectangle
right
rounded
roundedRectangle
scrollBar
scrolling
second
seventh
shadowed
singleLineWrap
singleSelect
sixth
solidFill
solidStroke
spectrumPalette
statusBox
strikeout
system
tenth
text
textLine
textLines
the
third
this
time
to
toolPalette
top
true
underline
vertical
white
with
word
words
wordWrap

a, an

Special term

Description

Used with the draw command to make statements easier to read. These terms are optional in every context in which they can be used.

Example

```
draw an arc from 2000,2500 to 4500,4500 to 3440,4000
```

after

Special term

Description

Used with the following OpenScript commands to indicate that a value should follow other values:

pop
put

Example

```
put sysDate after text of field "Contacts"
   -- Appends the current date to the field "Contact"
```

again

Special term

Description

Used with the `search` command to indicate that ToolBook should search again for specified text.

Example

```
search again
```

alias

Special Term

Description

Optionally used with the add menuItem command to specify the message to be sent when a user chooses a user-defined menu command.

Example

```
add menuItem "Addresses" alias "myAddress" to menu "File"
   -- adds a menu item, "Addresses", to the file menu. When
   -- the item is selected the message, "myAddress", will be
   -- sent to the current page.
```

all

Special term

Description

Indicates everything of a particular type or everything within a particular scope.

Examples

```
flip all
print all pages
select all recordField
```

angledLine

Special term

Description

The object type name of objects created with the angled line tool, the name of the tool itself, or the term for more than one angled line. You can use `angledLine` in any context in which it is appropriate as part of an object identifier.

Examples

```
draw with angledLine tool from 1000, 1000 to\
   2050, 2000 to 3050, 3000

hide angledLine "xy"

select all angledLine
```

arc

Special term

Description

The object type name of objects created with the arc tool, the name of the tool itself, or the term for more than one arc. You can use `arc` in any context in which it is appropriate as part of an object identifier.

Examples

```
draw an arc from 2000,2500 to 2000,4500 to 3440,4000

show arc "xyz"

select all arc
```

ascending

Special term

Description

Used with the `sort` command to indicate that ToolBook should sort values from the lowest ANSI value to the highest ANSI value.

Example

```
sort pages 1 to 3 by ascending text \
   text of recordField "Name"
   -- sorts pages 1 to 3 in ascending order according
   -- to the "Name" field
```

at

Special term

Description

A preposition that precedes terms that indicate a working level or a screen or page location.

Examples

```
activate menuItem "Books" at author
   -- Activates the Books command at Author level

add menu "&Lessons" at both

magnify 2 at 2400,3650
   -- Magnifies an area whose center is at 2400,3650

show the patternPalette at 1000,1400
```

author

Special term

Description

Used with the following OpenScript keywords to indicate the Author working level:

```
activate menuItem
add menu
add menuItem
check menuItem
deactivate menuItem
menuState
remove menu
remove menuItem
restore menuBar
sysLevel
uncheck menuItem
```

Scripts that set `sysLevel` to `author` will cause an error in Runtime ToolBook.

Examples

```
activate menuItem "Books" at author
   -- Activates the Books command at Author level

uncheck menuItem "Introduction" at author
   -- Unchecks the Introduction command at Author level
```

background

Special term

Description

The object type name of a background. You can use `background` in any context in which it is appropriate as part of an object identifier.

Example

```
send buttonDoubleClick to background id 1
```

before

Special term

Description

Used with the following OpenScript commands to indicate that one value should precede other values:

pop
put

Examples

```
put sysDate before it
      -- Prepends the current date to the text in It

put "not" before word 5 of text of field id 3 of page 7
      -- Puts the word not into the text of field id 3
      -- preceding the fifth word
```

black

Special term

Description

Used with the following commands to indicate the color black:

fxDissolve
fxWipe
fxZoom

Unlike the constant `black`, the special term `black` does not evaluate to `0,0,0`. Also see the constant `black`.

Example

```
fxDissolve fast to black
```

bold

Special term

Description

Used with the following properties to indicate bold type style:

fontStyle

sysFontStyle

Example

```
set sysFontStyle to bold
```

book

Special term

Description

The object type name of a book. You can use `book` in any context in which it is required or acceptable as part of an object identifier. You can also use `book` with the following commands to indicate a book:

```
edit script
forward
go
save changes
send
```

Examples

```
get the pageCount of this book

go to book "c:\mybooks\contacts.tbk"
   -- Puts the focus in the first page of contacts.tbk

save changes to book "c:\toolbook\newbook.tbk"
```

both

Special term

Description

Used with the following OpenScript keywords to indicate `Author` and `Reader` working levels:

```
activate menuItem
add menu
add menuItem
check menuItem
deactivate menuItem
menuState
remove menu
remove menuItem
restore menuBar
uncheck menuItem
```

Example

```
add menuItem "List" to menu "Books" at both
```

bottom

Special term

Description

Used to indicate the lower edge of an object on a page or the lower edge of a page.

Use of bottom in scripts

Used with the...	To indicate...
`align`	The lower edge of objects on a page
`fxWipe`	The lower edge of a page

Examples

```
align bottom
```

button

Special term

Description

The object type name of objects created with the button tool, the name of the tool itself, or the term for more than one button. You can use `button` in any context in which it is appropriate as part of an object identifier.

Examples

```
edit script of button "Next"
move button "Index" by 1440,1440
```

by

Special term

Description

Precedes terms that indicate amounts or other specifications.

Use of by in scripts

Used with ...	To precede...
decrement	An amount by which to decrement
increment	An amount by which to increment
move	An amount by which to move
search	The term `case`, which indicates to search by the case of the letters in the search screen
sort	The sort keys by which to search
step	An amount by which a variable should be incremented or decremented

Example

```
sort pages 1 to 3 by \
   descending text text of recordField id 0
   -- Does a text-based descending sort of pages 1 through 3
   -- based on the contents of record field id 0 on the
   -- current page
```

center

Special term

Description

Used with the following properties to indicate centered text alignment:

sysAlignment
textAlignment

Example

```
set textAlignment of the selection to center
```

char, character, characters, chars

Special term

Description

The special terms `char` and `character` indicate one character of text. The special terms `chars` and `characters` indicate one or more characters of text. You can use these special terms in any context in which a string specifier expression is appropriate. For details, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
get chars 10 to 12 of textLine 3 of text of field "Address"
```

checkbox

Special term

Description

Used with the `borderStyle` property to indicate the checkbox button style.

Example

```
set borderStyle of button "Edit" to checkbox
   -- Sets the borderStyle for a selected button
```

colorTray

Special term

Description

Used with the following commands to indicate the color tray palette:

hide
move
show

Examples

```
show colorTray

move colorTray to 7200,0
```

columns

Special term

Description

Used with the `printerStyle` property to indicate ToolBook should print a column report.

Example

```
set printerStyle to columns
```

commandWindow

Special term

Description

Used with the following commands to indicate the Command window:

clear
hide
move
put
set
show

Examples

```
hide the commandWindow

put it into the commandWindow
   -- Puts the value in It into the Command window
```

curve

Special term

Description

The object type name of objects created with the curve tool, the name of the tool itself, or the term for more than one curve. You can use `curve` in any context in which it is appropriate as part of an object identifier.

Example

```
edit script of curve id 3
```

dashed

Special term

Description

Used with the following properties to indicate a dashed line:

lineStyle
sysLineStyle

Examples

```
set the sysLineStyle to dashed

put dashed into lineStyle
```

date

Special term

Description

Indicates that ToolBook should interpret a value as a date.

Use of date in scripts

Used with ...	To indicate...

`<`
`<=`
`<>`
`=`
`>`
`>=`
`is`
`is not`
	Operands should be compared as dates

`format`	To format dates
`sort`	To sort by dates

Examples

```
if textLine 1 of text of field "Last Date" <=\
   sysDate as date
   format date textline 1 of text of field id 1 as \
      "y mmm dd" from "mm/yy"
end
```

decimal

Special term

Description

Used with the following properties to indicate decimal aligned tabs:

sysTabType
tabType

Example

```
set sysTabType to decimal
```

descending

Special term

Description

Used with the `sort` command to indicate that ToolBook should sort values from the highest ANSI value to the lowest ANSI value.

Example

```
sort pages 1 to 3 by \
   descending text text of recordField "Topic"
```

dotted

Special term

Description

Used with the following properties to indicate a dotted line style:

lineStyle
sysLineStyle

Example

```
set the sysLineStyle to dotted
```

eighth

Special term

Description

Used in string specifier expressions to indicate the eighth character, word, item, or text line, and in object identifiers to indicate the eighth page of a book. For details, see "Expressions with String Specifiers" and "Referring to Objects" in Chapter 2, "Script Basics", of Using OpenScript.

Example

```
put the eighth word of text of field "Topics" into it
```

ellipse

Special term

Description

The object type name of objects created with the ellipse tool, the name of the tool itself, or the term for more than one ellipse. You can use `ellipse` in any context in which it is appropriate as part of an object identifier.

Examples

```
draw an ellipse from 600,1500 to 2100,2100

select all ellipse
```

end

Special term

Description

Used to end control structures and handlers.

Use of end in scripts

Used with ...	To...

`to get`
`to handle`
`to set`
	End the handler structure

`conditions/when/else`
`if/then/else`
`linkDLL`
`while`
`start spooler`
`step`
	End the control structure

To end a handler, you can use `end` by itself as a statement, or you can use `end` followed by the message, user-defined property, or user-defined function used in the first statement in the handler. For example, to end a handler that starts with `to handle buttonUp`, you can use `end` or `end buttonUp`.

To end a control structure, you can use `end` by itself or `end` followed by the term that starts the control structure. For example, to end an `if/then/else` control structure, you can use `end` or `end if`.

The `do/until` control structure ends with the `until` statement, not an `end` statement.

Example

```
to handle enterBook
   system testScore
   set testScore to 0
end enterBook
```

english

Special Term

Description

Used with the `sysUnits` property to indicate inches as the unit of measure for grid size, page size, tabs, and indents.

Example

```
set sysUnits to english
   -- sets unit of measure to inches
```

excluding

Special Term

Description

Precedes the `background` keyword when used with the search command to indicate that the background fields will not be searched.

Example

```
search excluding background for "Sarah"
   -- searches for "Sarah" excluding background fields
```

false

Special term

Description

The logical value false. The opposite of `true`.

Examples

```
set visible of button "Help" to false
   -- Makes the Help button invisible

set the sysGrid to false
   -- ToolBook hides the grid
```

field

Special term

Description

The object type name of objects created with the field tool, the name of the tool itself, or the term for more than one field. You can use `field` in any context in which it is appropriate as part of an object identifier.

Examples

```
get the text of field id 12; put it into the text of\
   field id 13

set the borderStyle of field "Help" to shadowed
```

fifth

Special term

Description

Used in string specifier expressions to indicate the fifth character, word, item, or text line, and in object identifiers to indicate the fifth page of a book. For details, see "Expressions with String Specifiers " and "Referring to Objects" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
select the fifth word of the text of field "Essay"
```

first

Special term

Description

Used in string specifier expressions to indicate the first character, word, item, or text line, and in object identifiers to indicate the first page of a book. For details, see "Expressions with String Specifiers" and "Referring to Objects" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
get the first word of the text of field "name"
```

for

Special term

Description

Precedes a search string or the number of characters to read from a file.

Use of for in scripts

Used with ...	To precede...
`readFile`	The number of characters that ToolBook should read
`search`	The characters for which ToolBook should search

Example

```
search records for "the" by case as word
   -- Finds and selects the first occurrence of the word
   -- "the" in lowercase in a record field
```

fourth

Special term

Description

Used in string specifier expressions to indicate the fourth character, word, item, or text line, and in object identifiers to indicate the fourth page of a book. For details, see "Expressions with String Specifiers" and "Referring to Objects" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
get the fourth item of my userProperties
```

from

Special term

Description

Precedes a starting value.

Use of from in scripts

Used with ...	To precede...
`draw`	The location from which to draw
`format`	The old format to reformat
`select all`	The location from which to select
`step`	The value from which to step

Examples

```
draw a rectangle from 1440,2000 to 3000,3000

format date textline 1 of text of field id 1 as "y mmm dd" \
   from "mm/yy"

select all from Beginning to Finish

step i from 1 to pageCount of this book
   put name of page i
end
```

gray

Special term

Description

Used with the following commands to indicate the color gray:

fxDissolve
fxWipe
fxZoom

Example

```
fxDissolve to gray
```

group

Special term

Description

The object type name for a group or the term for more than one group. You can use `group` in any context in which it is appropriate as part of an object identifier.

Examples

```
hide group "answers"

select all group
```

groups

Special term

Description

Used with the printerStyle property to indicate ToolBook should print a group report.

Example

```
set printerStyle to groups
```

horizontal

Special term

Description

Used with the `align` command to indicate ToolBook should align objects on a page along their horizontal center.

Example

```
align horizontal
```

hotword

Special term

Description

The object type name for a hotword. You can use `hotword` in any context in which an object identifier for a hotword is applicable.

Example

```
get the text of hotword id 10
```

id

Special term

Description

Used in an object identifier to precede the ID number.

Examples

```
put text of hotword id 3 into the commandWindow
```

```
move irregularPolygon id 5 to 1440,1440
```

inches

Special term

Description

This term is obsolete in ToolBook version 1.5. See english

into

Special term

Description

Used with the following commands to precede a reference to a destination stack or container:

put
pop

Examples

In the following example, if field `Test Question 4` contains the expression `average(20,30,40)`:

```
put text of field "Test Question 4" into it
   -- Puts average(20,30,40) into It

put sysTime into text of field "Current Time"
   -- Puts the system time into the field "Current Time"
```

irregularPolygon

Special term

Description

The object type name of objects created with the irregular polygon tool, the name of the tool itself, and the term for more than one irregular polygon. You can use `irregularPolygon` in any context in which it is appropriate as part of an object identifier.

Examples

```
edit script of irregularpolygon id 3

select all irregularPolygon
```

italic

Special term

Description

Used with the following properties to indicate italic type style:

fontStyle
sysFontStyle

Example

```
set my fontStyle to italic
```

item, items

Special term

Description

The special term `item` means one value in a comma-separated list. If there is a space before the value, it is included as part of the value. The comma is not included. The special term `items` means more than one value in a comma-separated list. You can use these special terms in any context in which a string specifier expression is appropriate. For details, see "Expressions with String Specifiers" in Chapter 2 of Using Openscript, "Script Basics."

Example

```
put item 1 of varList into the commandWindow
```

justify

Special term

Description

Used with the following properties to indicate ToolBook should align text along the left and right margins:

sysAlignment
textAlignment

Example

```
set my textAlignment to justify
```

last

Special term

Description

The last page of a book, the last character in a word or item, the last item in a list, the last word in a text line, or the last text line in a field. You can use `last` in any context in which a string specifier expression is appropriate, and in object identifiers to indicate the last page of a book. For details, see "Expressions with String Specifiers" and "Referring to Objects" in Chapter 2, "Script Basics," in Using OpenScript.

Examples

```
format time last textline of footer of this book
   -- Formats the last line of the footer as sysTimeFormat

go to last page

put yourName into the last word of textline 4 of the \
   script of hotword id 7
```

left

Special term

Description

The left edge of an object on a page, the left edge of a page, or the left margin of a field.

Use of left in scripts

Used with ...	To indicate...

`sysAlignment`
`textAlignment`
`sysTabType`
`tabType`
	The left margin of a field

`align`	The left edge of object on a page
`fxWipe`	The left edge of a page

Examples

```
align left
   -- Aligns the selected objects along their left edges

fxWipe left fast to next page
   -- Moves the next page onto the screen from the right
```

line

Special term

Description

The object type name of objects created with the line tool, the name of the tool itself, and the term for more than one line. You can use `line` in any context in which it is appropriate as part of an object identifier.

Example

```
draw a line from 233,1500 to 233,2450
```

linePalette

Special term

Description

Used with the following OpenScript keywords to indicate the line palette:

```
bounds
hide
move
position
show
```

Examples

```
set the position of the linePalette to 0,0

move the linePalette by 1440,1440

get the bounds of the linePalette
```

locateOnly

Special term

Description

Used with the `search` command to search for text without selecting it or navigating to the page that contains the text. Instead, the content of It is set to a list of three items: the name of the field or record field where the text was found, followed by two integers that indicate the character range of the matching text in that field.

Example

```
search for "WA" locateOnly
   -- searches for "WA" and returns location of found string
   -- in It
```

mainWindow

Special term

Description

Used with the following OpenScript keywords to indicate the main ToolBook window:

```
bounds
hide
move
position
show
```

Examples

```
get the bounds of the mainWindow

hide the mainWindow
```

menuBar

Special term

Description

Used with the following commands to indicate the main ToolBook window menu bar:

hide
restore menuBar
show

Examples

```
hide the menuBar

show menuBar
```

metric

Special term

Description

Used with and sysUnits property to indicate centimeters as the unit of measure for grid and size, tabs, and indents.

Example

```
set sysUnits to metric
   -- sets unit of measure to centimeters
```

mid, middle

Special term

Description

The middle character, word, item, or text line. You can use `mid` and `middle` in any context in which a string specifier expression is appropriate. If there are an even number of items (characters, words, list items, or textLines) then `middle` will return the item just to the left of middle (e.g. the result of `middle of "1, 2, 3, 4"` will be 2.)

For details on expressions, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
get the middle textLine of text of \
   field "Donations by Calendar Date"
```

millimeters

Special term

Description

This term is obsolete in ToolBook version 1.5. See metric.

multiSelect

Special term

Description

A value of the `fieldType` field and recordField property. If `fieldType` has the value, `multiSelect`, the field or recordField will behave as a multiple-choice list box. After setting a field's `fieldType` property to `multiSelect`, when a reader clicks a textline in the field, the textline will be selected. Clicking additional textlines will cause them to be selected without unselecting previous lines. The value of the field's `selectedTextLines` property will be a list of textline numbers of the selected textlines.

Examples

```
set fieldType of field "field list" to multiSelect
   -- makes field "file list" a multiSelect list box
```

my

Special term

Description

Used in statements that refer to properties to indicate that the properties belong to the object whose script is currently executing.

Examples

```
get my header
   -- Gets the value of the header property for the book
   -- whose script is currently executing

set my position to 1500,3000
```

name

Special term

Description

Indicates that ToolBook should interpret a value as a name. When ToolBook compares values as names, ToolBook uses only the first paragraph of a string in a container in the comparison. First, ToolBook compares last names by looking for the third word of the string in each container. If there is not a third word, ToolBook looks for the second word. If there is not a second word, ToolBook looks for the first word. In this way, ToolBook compares last names first. If a container is empty, the string is compared as a null string, which has a greater value than any other string.

After comparing last names, ToolBook compares first names by comparing the first word of the string in each container. If there is only one word in the string, first names are not compared. If there are three words in each string, ToolBook compares the second word of each string as middle names.

Use of name in scripts

Used with ...	To indicate...

`<`
`<=`
`<>`
`=`
`>`
`>=`
`is`
`is not`
	Operands should be compared as names

`sort`	To sort as names, as opposed to numbers, text, or dates

Example

```
if text of field "Test 1" > text of field "Test 2" as name
   send switch field "Test 1", field "Test 2"
end if
```

next

Special term

Description

Used in object identifiers to refer to the page after the current page.

Example

```
go to next page
```

ninth

Special term

Description

Used in string specifier expressions to indicate the ninth character, word, item, or text line, and in object identifiers to indicate the ninth page of a book. For details, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
go to the ninth page of this book
```

none

Special term

Description

Used with the following properties to indicate the absence of other values for the property:

borderStyle
lineStyle
pattern
sysCursor
sysLineStyle

Example

```
-- If there is no visible line around the graphic
if my lineStyle is none
   set my lineStyle to 4
end if
```

noWrap

Special term

Description

Used with the `fieldType` property to indicate that the field displays multiple textlines without word wrap.

Example

```
set fieldType of field "Choices" to noWrap
```

number

Special term

Description

Indicates that ToolBook should interpret a value as a number.

Use of number in scripts

Used with ...	To indicate...

`<`
`<=`
`<>`
`=`
`>`
`>=`
`is`
`is not`
	Operands should be compared as numbers

`format`	To format numbers
`sort`	To sort by numbers

Examples

```
format number text of field "Tax" of page 2

if text of field "Contacts" is 25 as number
   go to page "Bonus Points"
end if
```

of

Special term

Description

Indicates hierarchical relationships in string specifier expressions and object identifiers. The special term `of` can be used in any context in which a string specifier expression or object identifier is appropriate. For details, see "Expressions with String Specifiers" in Chapter 2 in Using OpenScript, "Script Basics."

This special term is also used to indicate that a property belongs to a particular object.

Example

```
get the first textLine of text of recordField\
   "Definition" of page 5
   -- The first of is part of a string specifier, the second
   -- of indicates that the text property belongs to a
   -- particular record field, and the third of is part of
   -- the object identifier for that record field
```

onto

Special term

Description

Precedes a reference to a destination stack in a `push` statement.

Example

```
push sysDate onto textLine 1 of text of field "Access_dates"
```

page

Special term

Description

Used with the following commands as the object type name for a page:

edit script
forward
fxDissolve
fxWipe
fxZoom
go
send

You can use `page` in any context in which it is appropriate as part of an object identifier.

Example

```
set text of field "trees" of page 7 to "evergreens"
```

pages

Special term

Description

Used with the following OpenScript keywords to indicate pages in a ToolBook book:

```
flip
print
sort
printerStyle
```

Examples

```
flip 5 pages

print all pages

sort pages 1 to 3 by \
   descending text text of recordField "Name"
   -- Does a text-based descending sort of
   -- pages 1 through 3 based on the contents of
   -- recordfield "Name"
```

paintObject

Special term

Description

The object type name for paint objects, and the term for more than one paint object. You can use `paintObject` in any context in which it is appropriate as part of an object identifier.

Example

```
move paintObject "Palm Tree" to 5000,1000
```

patternPalette

Special term

Description

Used with the following OpenScript keywords to indicate the pattern palette:

```
bounds
hide
move
position
show
```

Examples

```
show the patternPalette

move the patternPalette to 3000,0
```

picture

Special term

Description

The object type name of an object created by pasting a Windows metafile or importing files that have the file name extensions .WMF, .DRW, .EPS, .CGM, or .TIF. You can use picture in any context for which it is appropriate as part of an object identifier.

Example

```
set the position of picture "Globe" to 10,10
```

pie

Special term

Description

The object type name for objects created with the pie tool, the name of the tool itself, or a term for more than one pie. You can use `pie` in any context in which it is appropriate as part of an object identifier.

Examples

```
send buttonUp to pie id 8

move pie id 8 to 4000,4000
```

polygon

Special term

Description

The object type name of objects created with the polygon tool, the name of the tool itself, or the term for more than one polygon. You can use `polygon` in any context in which it is appropriate as part of an object identifier.

Examples

```
select polygon "Box"

edit script of polygon id 3
```

polygonPalette

Special term

Description

Used with the following OpenScript keywords to indicate the polygon palette:

```
bounds
hide
move
position
show
```

Examples

```
hide the polygonPalette
```

```
get the position of the polygonPalette
```

previous

Special term

Description

Used in object identifiers to indicate the page before the current page.

Example

```
go to previous page
```

pushButton

Special term

Description

Used with the button `borderStyle` property to indicate that the buttons appears three-dimensional, like Windows 3.0 dialog box buttons. When the value of a button's `borderStyle` property is `pushButton`, you can also set the button's `invert` property to true to make the button appear as though it is being pushed.

Example

```
set borderStyle of button "stop" to pushButton
```

radioButton

Special term

Description

Used with the `borderStyle` property to indicate the radio button style for a button.

Example

```
set my borderStyle to radioButton
```

reader

Special term

Description

Used with the following OpenScript keywords to indicate the Reader working level:

```
activate menuItem
add menu
add menuItem
check menuItem
deactivate menuItem
menuState
remove menu
remove menuItem
restore menuBar
sysLevel
uncheck menuItem
```

Examples

```
set sysLevel to reader

check menuItem "Lessons" at reader
```

recordField

Special term

Description

The object type name for objects created with the record field tool, the name of the tool itself, or the term for more than one record field. You can use `recordField` in any context in which it is appropriate as part of an object identifier.

Examples

```
draw a recordField from 2560,2560 to 5000,5000

set the name of recordField id 6 to "Address"
```

rectangle

Special term

Description

The object type name of objects created with the rectangle tool, the name of the tool itself, the term for more than one rectangle, or the term for the rectangle button or field style. You can use `rectangle` in any context in which it is appropriate as part of an object identifier.

Examples

```
show rectangle "Picture"

select all rectangle
```

right

Special term

Description

Used to indicate right edge of objects on a page, the right edge of a page, or the right margin of a field.

Use of right in scripts

Used with ...	To indicate...

`sysAlignment`
`textAlignment`
`sysTabType`
`tabType`
	The right margin of a field

`align`	The right edge of objects on a page
`fxWipe`	The right edge of a page

Examples

```
align right
   -- Aligns the selected objects along their right edges

fxWipe right fast to next page
   -- Moves the next page onto the screen from the right
```

rounded

Special term

Description

Used with the `borderStyle` property to indicate the rounded rectangle button style.

Example

```
set my borderStyle to rounded
```

roundedRectangle

Special term

Description

The object type name of objects created with the rounded rectangle tool, the name of the tool itself, or the term for more than one rounded rectangle. You can use `roundedRectangle` in any context in which it is appropriate as part of an object identifier.

Example

```
draw a roundedRectangle from 3000,3000 to 4440,4440
```

scrollBar

Special term

Description

Used with the following commands to indicate the scroll bar:

```
hide
show
```

Example

```
hide the scrollBar
```

scrolling

Special term

Description

Used with the `borderStyle` property to indicate the scrolling field style.

Example

```
set my borderStyle to scrolling
```

second

Special term

Description

Used in string specifier expressions to indicate the second character, word, item, or text line, and in object identifiers to indicate the second page of a book. For details, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
go to the second page of book "c:\ToolBook\datebook.tbk"
```

seventh

Special term

Description

Used in string specifier expressions to indicate the seventh character, word, item, or text line, and in object identifiers to indicate the seventh page of a book. For details, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
select seventh textLine of text of field "Booklist"
```

shadowed

Special term

Description

Used with the `borderStyle` property to indicate the shadowed button style.

Example

```
set the borderStyle of button "Index" to shadowed
```

singleLineWrap

Special term

Description

Used with the fieldType property to indicate that the field displays only a single line of text without word wrap.

Example

```
set the fieldType of field "Name" to singleLineWrap
   -- only allows a single line in field "Name"
```

singleSelect

Special term

Description

Used with the `fieldType` property to indicate that the field behaves as a singleSelect list box. After setting a field's `fieldType` property to `singleSelect`, when a reader clicks a textline in the field, the textline will be selected and the value of the field's `selectedTextLines` property will be the textline number.

Example

```
set the fieldType of field "Name List" to singleSelect
   -- Causes field "Name List" to behave as a single-choice
   -- listbox.
```

sixth

Special term

Description

Used in string specifier expressions to indicate the sixth character, word, item, or text line, and in object identifiers to indicate the sixth page of a book. For details, see "Expressions with String Specifiers" and "Referring to Objects" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
get the sixth char of the text of field "Grades"
```

solidFill

Special term

Description

Used with the following properties to indicate that an object’s pattern is the solid fill color:

pattern
sysPattern

Example

```
set sysPattern to solidFill
```

solidStroke

Special term

Description

Used with the following properties to indicate that an object’s pattern is the solid stroke color:

pattern
sysPattern

Example

```
set pattern of rectangle "Costs" to solidStroke
```

spectrumPalette

Special term

Description

Used with the following commands to indicate the spectrum palette:

hide
move
show

Examples

```
show spectrumPalette

move spectrumPalette to 0,4320
```

statusBox

Special term

Description

Used with the following commands to indicate the status box:

hide
show

Example

```
show statusBox
```

strikeout

Special term

Description

Used with the following properties to indicate strikeout type style:

fontStyle
sysFontStyle

Example

```
set the fontStyle of selectedText to strikeout
```

system

Special term

Description

Used to indicate the ToolBook system, the system font typeface, or any instance of a particular Windows application.

Use of system in scripts

Used with ...	To indicate...

`forward`
`break`
`send`
	The ToolBook system

`executeRemote`
 	
`getRemote`
	Any instance of a particular Windows application

`sysFontFace`	The system font typeface

Examples

```
break to system

send open to system

forward to system
```

tenth

Special term

Description

Used in string specifier expressions to indicate the tenth character, word, item, or text line, and in object identifiers to indicate the tenth page of a book. For details, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
get the tenth item of sysHistory
```

text

Special term

Description

Indicates that ToolBook should read particular values as text.

Use of text in scripts

Used with ...	To indicate...

`<`
`<=`
`=`
`>`
`>=`
`is`
`is not`
	Operands should be compared as text

`sort`	To sort by text

Example

```
if text of field "Name" > "K" as text
   go page "L through Z"
end if
```

textLine, textLines

Special term

Description

The special term `textLine` refers to a string that ends with a Carriage Return/Line Feed (`crlf`) character. The special term `textLines` refers to one or more strings that end with `crlf`.

Note that the end of a text line in a field is not necessarily where word wrap occurs, but where a `crlf` is entered.

You can use these special terms in any context in which a string specifier expression is appropriate. For details, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," in Using OpenScript.

Example

```
get the first textLine of the text of field "Booklist"
```

the

Special term

Description

Used in statements to make them easier to read. It is optional in every context in which it is used.

Example

```
set the highlight of button "Next" to true
```

third

Special term

Description

Used in string specifier expressions to indicate the third character, word, item, or text line, and in object identifiers to indicate the third page of a book. For details, see "Expressions with String Specifiers" and "Referring to Objects" in Chapter 2, "Script Basics," in Using OpenScript.

Examples

```
go to the third page of this book

put the third textLine of my script into commandWindow
```

this

Special term

Description

Used with the object identifier for a page, background, or book to indicate the current page, background, or book.

Examples

```
get the customColors of this book
get the objects of this background
```

time

Special term

Description

Used with the `format` command to indicate ToolBook should format a string to the specified time format.

Example

```
Format time text of field "AccessTime" as "hh:mm" from\
   "seconds"
```

to

Special term

Description

Used to precede a location, container, or desired value.

Use of to in scripts

Used with ...	To precede the...
`add menuItem`	Menu to add a command to
`draw`	Location to draw to
`break` `forward`	ToolBook system
`fxDissolve`	Result that the ToolBook display dissolves to
`fxWipe`	Result that the ToolBook display wipes to
`fxZoom`	Result that the ToolBook display zooms to
`go`	Object to go to
`move`	Location to move an object, window, or palette to
`readFile`	Character to read a file to
`save changes`	Book to save changes to
`select all`	Location to select to
`send`	Object to send a message to
`set`	Value to set a container to
`sort`	Page to sort values to
`step`	Value to step a variable to
`writeFile`	File to write text to

Examples

```
draw from 1025,1025 to 2422,2021

fxWipe left fast to next page
   -- Moves the next page onto the screen from right to
   -- left

go to book "c:\mybooks\contacts.tbk"
   -- Puts the focus in the first page of contacts.tbk

go to page 5 of book "lessons.tbk"

step i from 1 to pageCount of this book
   put name of page
end
```

```
writeFile text of field "Name" of page 1 of this book to\
   "newdata.dat"
   -- Writes text to the file
```

toolPalette

Special term

Description

Used with the following OpenScript keywords to indicate the tool palette:

```
bounds
hide
move
position
show
```

Examples

```
get the bounds of the toolPalette
```

```
hide the toolPalette
```

top

Special term

Description

Used to indicate the upper edge of an object on a page or the upper edge of a page.

Use of top in scripts

Used with ...	To indicate...
`align`	The upper edge of objects on a page
`fxWipe`	The upper edge of a page

Examples

```
align top
```

```
fxWipe top fast to the next page
```

true

Special term

Description

The logical value true. The opposite of `false`.

Examples

```
set my highlight to true
```

```
send buttonUp 3000,3000,true,false
```

underline

Special term

Description

Used with the following properties to indicate underline type style:

```
fontStyle
sysFontStyle
```

Examples

```
set fontStyle of recordField "Title" of book "articles.tbk"\
   to underline

set the sysFontStyle to underline
```

vertical

Special term

Description

Used with the `align` command to indicate the vertical center of objects on a page.

Example

```
align vertical
```

white

Special term

Description

Used with the following commands to indicate the color white:

fxDissolve
fxWipe
fxZoom

Unlike the constant `white`, the special term `white` does not evaluate to `0,100,0`. Also see the constant `white` in this chapter.

Examples

```
fxDissolve fast to white

fxWipe left fast to white
```

with

Special term

Description

Precedes replies to questions in `ask` and `request` statements, and precedes a tool name in `draw` statements.

Use of with in scripts

Used with ...	To precede...
`ask`	The default answer
`draw`	The tool with which to draw
`request`	The replies to a request

Examples

```
ask "Who was the first Roman emperor?" with "Julius Caesar"
```

```
draw with line from 300,3500 to 6000,4000
```

word, words

Special term

Description

The special term `word` is used to refer to one word in a expression. The special term `words` means one or more words in an expression. A word is defined as:

- A sequence of printable, non-space characters delimited by the beginning of the expression and by the last character in the expression or by a space or non-printable character
- A sequence of printable, non-space characters immediately preceded by a space or non-printable character and terminated by the last character of the expression or by a space or non-printable character

You can use these special terms in any context in which a string specifier is appropriate. For details, see "Expressions with String Specifiers" in Chapter 2, "Script Basics" of Using OpenScript.

Examples

```
select word 3 of text of field "Names"

get words 3 to 4 of text of field "Address"
```

wordWrap

Special term

Description

Used with the field `fieldType` property to indicate that the field text wraps at the right margin.

Example

```
set the fieldType of field "Information" to wordWrap
```

Special Variables

Quick reference

argCount
argList
it

argCount

Special variable

Description

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

The `argCount` variable is most often used to process arguments passed with userdefined functions, properties, and messages. The value of `argCount` is not necessarily the same as the value returned by `itemCount(argList)` because a parameter can contain more than one item, such as the comma-separated list of coordinates that can make up a `<location>` parameter.

Value

The value is a non-negative integer equal to the number of parameters passed to the current handler. This integer defines the valid range of operands to the `argument` operator.

Example

```
to handle checkit num1, num2, num3, num4
   -- Handles a user-defined message
   if argCount < 4 then
      -- Check to make sure four arguments
      -- are passed.; if not, request numbers
      -- and exit the handler
      request "Please enter four numbers in the field."
      break
   else
      -- If there are the correct number of arguments,
      -- verify the numbers
      request "Are these numbers correct?" & crlf & argList \
         with "Yes" or "No"
      if "Yes" is in It
         send process argList to page 1
            -- Sends a user-defined process
            -- message with the argument list
      else
         -- If reader doesn't like the displayed numbers,
         -- request new numbers and exit the handler
         request "Please enter four numbers separated by commas"
         break
      end if
   end if
end checkit
```

argList

Special variable

Description

A special local variable that contains a comma-separated list of the parameters of the currently executing handler.

Value

The value is a list of all the parameters passed to the current handler, with a comma between each parameter. If ToolBook didn’t pass any parameters to the current handler, the value is null.

Example

```
to handle checkit num1, num2, num3, num4
   -- Handles a user-defined message
   if argCount < 4 then
      -- Check to make sure four arguments
      -- are passed; if not, request numbers
      -- and exit the handler
      request "Please enter four numbers in the field."
      break
   else
      -- If there are the correct number of arguments,
      -- verify the numbers
      request "Are these numbers correct?" & crlf & argList \
         with "Yes" or "No"
      if "Yes" is in It
         send process argList to page 1
            -- Sends a user-defined process
            -- message with the argument list
      else
         -- If reader doesn't like the displayed numbers,
         -- request new numbers and exit the handler
         request "Please enter four numbers separated by commas"
         break
      end if
   end if
end checkit
```

It

Special variable

Description

`It` is a special local variable. You don't have to declare this variable, and it is available as a local variable in all handlers. Like a local variable, 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 a `get`, `ask`, or `request` command into `It`.

Keep in mind that the data put into `It` by one command will be overwritten by the data put into `It` by another command. As a rule, if you need a value that is returned in `It` for more than just the next statement in a handler, you should put the value of `It` into another variable where the value will not be overwritten. For details about variables, see "Variables" in Chapter 2, "Script Basics," in Using OpenScript.

Examples

```
get text of field "Washington"
   -- Gets the contents of field Washington and
   -- puts the contents into It

put it into commandWindow
   -- Puts the contents of It into the Command window

ask "What is your answer?"
put it into Answer
   -- Stores the answer in the Answer variable for use
   -- later in the handler
```

Commands

Quick reference

Alphabetical List

Menu control

activate menuItem
add menu
add menuItem
check menuItem
deactivate menuItem
remove menu
remove menuItem
restore menuBar
uncheck menuItem

User interface and navigation

ask
ask password
beep
flip
flip all
fxDissolve
fxWipe
fxZoom
go
magnify
request
search

Image Control

remove backgroundImage
remove pageImage
store backgroundImage
store pageImage

Variable declaration

local
system

Data and object manipulation

align
clear
decrement
draw
extend select
format
get
hide
increment
move
pop
push
put
return
seed
select
set
show
sort
unselect

File manipulation and print commands

closeFile
createFile
openFile
print
print eject
readFile
save as
save changes
writeFile

Script control

break
break to system
continue
edit script
execute
forward
pause
restore system
return
send

DDE, DLL, and data exchange

closeRemote
executeRemote
export
getRemote
import
importGraphic
keepRemote
respondRemote
run
setRemote
unlinkDLL
untranslateAllWindowMessages
untranslateWindowMessage

Commands: Alphabetical list

Quick reference

```
activate menuItem <menu item> [at <level>]
add menu <menu name> [position <position>] [at <level>]
add menuItem <menu item> alias <message> to menu <menu name>
   [position <position>] [at <level>]
align <type>
ask <question> [with <default answer>]
ask password <question>

beep <number>
break [<handler name>]
break <control structure>
break to system

check menuItem <menu item> [at <level>]
clear <container>
clear <object>
closeFile <file name>
closeRemote [applicaton <server name>] [topic <topic name>]
continue [<control structure>]
createFile <file name>

deactivate menuItem <menu item> [at <level>]
decrement <expression> [by <amount>]
draw [with | a | an] <tool> [tool] from <location> to <location>
   [to <location>...]

edit [the] script of <object>
execute <source>
executeRemote <command> [application <server name>] [topic
   <topic>]
export <file name> as <type> using <fields>

flip [<number> [pages]]
flip all [pages]
format [number] <container> [as <new format>] [from <old format>]
format date <container> [as <new format>] [from <old format>]
format time <container>  [as <new format>] [from <old format>]
forward <message> [<parameters>]
forward to system
fxDissolve [<speed>] [to <result>]
fxWipe <direction> [<speed>] [to <result>]
fxZoom [<speed>] [to <result>] [at <location>]

get<expression>
getRemote <data> [application <server name>] [topic <topic>]
go [to] <expression>

hide [<object>]

import <file name> as <type> using <fields>
importGraphic <file Name>
increment <expression> [by <amount>]
```

keepRemote [applicaton <server name>] [topic <topic name>]

local [variable[s]] <variable list>

magnify <power> [at <location>]
move [<object>] by <amount>
move [<object>] to <location>

openFile <file name>

pause <time> seconds
pause <time> [ticks]
pop [<stack>] [into | before | after] [<destination>]
print [<pages> [pages]]
print eject
push [<expression>] [onto <stack>]
put <expression>
put <expression> after <destination>
put <expression> before <destination>
put <expression> into <destination>

readFile <file name> for <number of characters>
readFile <file name> to <character>
remove backgroundImage | pageImage
remove menu <menu name> [at <level>]
remove menuItem <menu item> [at <level>]
request <question> [with <reply> [or <reply2> [or <reply3>]]]
respondRemote <response>
restore menuBar [at <level>]
restore system
return <expression>
run <application> [minimized]
run <file name> [minimized]

save as <file name>,<overwrite>
save changes to book <book name>
search again
search [page] for <string> [by case] [as word] [locateOnly]
search [page] excluding background for <string> [by case] [as word] [locateOnly]
search [page] records for <string> [by case] [as word] [locateOnly]
search [page] in <rfield> [, <rfield>] ... for <string> [by case] [as word] [locateOnly]
search [page] for <string> [by case] [as word]
seed <number>
[extend] select <objects>
select <string specifier>
select all [<object type>]
select all from <location> to <location>
send <message> [<parameters>] [to <object>]
set [the] <container> [of <object>] to <value>
setRemote <item> to <value> [application <server name>] [topic <topic>]
show [<object>] [at <location>]
sort [pages <number> to <number>] [by <sort key> [,<sort key>...]]
store backgroundImage | pageImage

system [variable[s]] <variable list>

uncheck menuItem <menu item> [at <level>]
unselect [<object>]
untranslateAllWindowMessages [for <winHandle>]
untranslateWindowMessage <winMsg> [for <winHandle>]

writefile <string> to <file name>

activate menuItem

Command

Syntax

```
activate menuItem <menu item> [at <level>]
```

Description

Reactivates a specified menu command if it was previously deactivated (dimmed). If a level is specified, the command is activated only at that working level (`Reader`, `Author`, or `both`). If a level is not specified, the command is activated at the current working level.

Changing the state of a default ToolBook menu command has no effect on the command.

Parameters

The `<menu item>` parameter is the name of a command as it appears on a menu. If you want, you can omit leading numerals and any non-alphanumeric characters (such as the ampersand, spaces and punctuation, but not underscores) so that the name matches the message that ToolBook sends when a user chooses the menu item. If you have defined the menu item with an alias using the add menuItem command, you can also use the alias name.

The `<level>` parameter must be `author`, `reader`, or `both`; the default is the working level in effect when the command is executed.

Examples

```
activate menuItem "Books" at author
   -- Activates the Books command at Author level

activate menuItem "advanced topics..."
   -- Activates the "Advanced Topics..." command at
   -- the current working level
```

add menu Command

Syntax

```
add menu <menu name> [position <position>] [at <level>]
```

Description

Adds a menu to the menu bar. If a position is given, the menu appears in that position on the menu bar. If a position is not given, ToolBook adds the menu to the right of all other menus except the Help menu. If a level is specified, the menu is added at that working level (Reader, Author, or both). If a level is not specified, the menu is added at the current working level when ToolBook executes the handler containing the `add menu` command.

Parameters

The `<menu name>` parameter is the name of a menu as you want it to appear on the menu bar. You must enclose the menu name in quotation marks if it contains spaces or punctuation. Also, the menu name must include at least one alphabetic character or underscore character (_). The menu name can be up to 60 characters long; however, good user-interface design should govern the lengths of the menu names you use.

To add a shortcut key, put an ampersand (`&`) in the menu name immediately preceding the character you want to use as the shortcut. If a shortcut key is defined, a user can choose the menu by pressing Alt plus the underlined character in the menu name, then choose a menu command from the menu by pressing the underlined character in the menu command name.

The `<position>` parameter is a positive integer that indicates the placement of the menu on the menu bar. For example, `position 3` means the third menu from the left on the menu bar. If there is already a menu in that position on the menu bar, that menu and all menus to its right shift to the right to accommodate the new menu.

The `<level>` parameter can be `author`, `reader`, or `both`. The default is the working level in effect when the command is executed.

Examples

```
add menu "Books" position 3
   -- Puts a menu called Books in the
   -- third position on the menu bar

add menu "&Books" position 3 at reader
   -- Puts a menu called Books in the third
   -- position on the menu bar at Reader level,
   -- with the B underlined as a shortcut key
```

add menuItem

Command

Syntax

```
add menuItem <menu item> [alias <message>] to menu <menu
   name>[position <position>][at level>]
```

Description

Adds a menu command to a menu. If a position is given, the command appears in that position on the menu. Otherwise, the command is added to the bottom of the menu. If a level is specified, the command is added at that working level (Reader, Author, or both). If a level is not specified, the command is added at the current working level. To add a separator bar to a menu, add a menu item that is a null string.

ToolBook sends a message to the current page when the command is chosen from the menu. For example, if you add a Functions command to the Edit menu, ToolBook sends a "functions" message to the current page when a user chooses that command. In order for anything to happen when a user selects the command, you must write a handler for the message and put it at the page level or higher in the object hierarchy.

Leading numerals and all non-alphanumeric characters other than the underscore (such as spaces and punctuation) are omitted from the message that is generated by choosing a command. For example, choosing the command "Enter Costs..." would send the message "entercosts".

If you include an alias, it specifies the message sent when the menuItem is selected in place of the menu item name. The fact that you can change the menu item without changing the message it sends (and therefore the name of the handler that responds to it) makes it easier to translate an application to work under versions of ToolBook localized for different languages (for example translating a book from English to French). When you remove a menu item with either the restore menubar command or the remove menuItem command, the alias for that menu item is also removed automatically.

Parameters

The `<menu item>` parameter is an expression that evaluates to a string that is the name of the menu command you want to add, including an ellipsis (. . .) if you want one. The name must include at least one alphabetic character or underscore character (_). You must enclose the command name in quotation marks if it contains spaces or punctuation. The command can be up to 60 characters long; however, good user-interface design should govern the lengths of the names of commands you add to menus.

The `<message>` parameter is any valid built-in or user-defined message. Valid messages are strings that begin with a letter and contain no spaces or punctuation marks except the underscore character.

The `<menu name>` parameter is an expression that evaluates to a string that is the name of a menu exactly as it appears on the menu bar. The menu name must be enclosed in quotation marks if it contains spaces or punctuation.

To add a shortcut key, put an ampersand (&) in the menu command name immediately preceding the character you want to use as the shortcut. The

shortcut key indicators look and act like the underlined characters on the standard ToolBook menus and menu commands.

The `<position>` parameter is a positive integer that indicates the placement of the command on the menu. For example, `position 3` means the third item in the menu. If there is already a command at that position in the menu, that command and all others below it shift down to accommodate the new command. If an invalid position is specified, ToolBook places the menu item either first or last on the menu, depending on which location is closer to the specified position.

The `<level>` parameter can be `author`, `reader`, or `both`; the default is the working level in effect when the command is executed.

Examples

```
add menuItem "List" to menu "Books" position 1
   -- Adds a List command to the top of a Books menu

add menuItem "&Calendar..." to menu "Books" at both
   -- Adds a "Calendar..." command with a shortcut key, C,
   -- to the bottom of the Books menu at Reader and Author
   -- levels

add menuItem "Search" alias "Find" to menu "Tools"
   -- Adds a "Search" command to the bottom of a Tools menu.
   -- When the menu item is chosen the message "Find" is
   -- sent to the current page.
```

align Command

Syntax

```
align <type>
```

Description

Aligns selected objects on their left, right, top, bottom, horizontal centers, or vertical centers, according to the type of alignment specified. Only one type of alignment can be specified. This command is equivalent to choosing Align from the Draw menu.

For left, right, top, or bottom alignment, the selected objects are aligned along their left, right, top, or bottom edges, respectively. For horizontal alignment, the selected objects are aligned on their horizontal centers. For vertical alignment, the objects are aligned on their vertical centers.

If no objects are selected, an error occurs and ToolBook displays the Execution Suspended message box.

Parameter

The `<type>` parameter must be `left`, `right`, `top`, `bottom`, `horizontal`, or `vertical`, or an expression that yields one of these values.

Examples

```
align top
   -- Aligns the selected objects along their top edges
```

In the following example, if field `Align` contains `horizontal`:

```
align text of field "Align"
   -- Aligns the selected objects horizontally
```

ask Command

Syntax

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

Description

Displays a dialog box containing a question, along with a text box into which the reader can type an answer. The dialog box contains OK and Cancel buttons. The optional `<default answer>` parameter specifies a string that appears initially in the text box, highlighted so that the reader can easily replace it. The answer goes into `It`, either when the reader clicks the OK button or presses Enter. If the reader clicks the Cancel button or presses Alt+F4 or Esc, ToolBook sets the `sysError` property to `Cancel` and the `sysErrorNumber` property to 566.

The `ask password` form of the command encrypts the answer.

Also see the `request` command.

Parameters

The `<question>` and `<default answer>` parameters are any expressions that yield strings.

Examples

```
ask "How many countries did the Romans conquer?"

ask "Who was the first Roman emperor?" with "Julius Caesar"

ask password "Please enter your password."

to handle buttonUp
   ask "Who is the most influential economist of this" &&\
      "century?"
      if sysError contains "cancel"
         go to page "Menu"
      end if
   conditions     -- Branch according to reader's response
      when it contains "Keynes"
         go to page "simple_keynesian_model"
      when it contains "Say"
         go to page "19th_century" of book "econrev.tbk"
      when it contains "Friedman"
         request "That's an interesting choice. Would" &&\
            "you like to read about him?" with "Yes" or "No"
         if it contains "Yes" then
            go to page "Late 20th century"
         else
            break
         end if
      else
         go to page "schools_of_thought"
   end conditions
```

```
end buttonUp
```

beep

Command

Syntax

```
beep <number>
```

Description

Causes the computer to beep. The number of times it beeps is determined by the value of the `<number>` parameter.

Parameter

The `<number>` parameter is any expression that yields a positive integer.

Example

```
conditions
   when answer is 3
      beep 30
   when answer is 4
      beep 40
   else
      beep 10
end conditions
```

break Command

Syntax

```
break [<handler name>]
break <control structure>
break to system
```

Description

Causes script execution to jump to the end of the current handler or control structure. The handler ends in the normal way, passing control back to the caller. The `break to system` form of the command causes control to pass directly to ToolBook, bypassing any other handlers.

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. This is because control can be passed from a `to get` handler to a calling handler only by means of the `return` statement.

Using `break` within an `if/then/else` control structure causes execution to jump to the end of the currently executing handler.

Parameters

The `<handler name>` parameter is the name of the current handler from which to exit.

The <control structure> parameter must be `do`, `step`, `conditions`, or `while`.

Example

```
to handle enterBook
   conditions
   when pageCount of this book is 1
      -- Exit the control structure when there is
      -- only one page in the book
      break conditions

   when pageCount of this book >1 and pageCount of \
      this book <= 4
      -- Tell the reader how many pages are in the book
      request "There are" && pageCount of this book && \
         "pages in this book."
   else
      request "There are enough pages in this book."
   end conditions
end enterBook
```

check menuItem

Command

Syntax

```
check menuItem <menu item> [at <level>]
```

Description

Puts a check to the left of a user-defined command name on a menu. If a level is specified, the command is checked at that working level (`Reader`, `Author`, or `both`). If a level is not specified, the menu item is checked at the working level in effect when the command is executed.

To remove a check from a menu command, use the `uncheck menuItem` command.

Changing the state of a default ToolBook menu command has no effect on the command.

Parameters

The `<menu item>` parameter is the name of a command as it appears on a menu. If you want, you can omit leading numerals and any non-alphanumeric characters (such as spaces and punctuation, but not underscores) so that the name matches the message that ToolBook sends when a user chooses the menu item.

The `<level>` parameter must be `author`, `reader`, or `both`.

Examples

```
check menuItem "Introduction"
   -- Puts a checkmark beside the Introduction command

check menuItem "Book List" at reader
   -- Puts a checkmark beside the Book List command at
   -- Reader level
```

clear

Command

Syntax

```
clear <object>
clear <container>
```

Description

Deletes an object or the contents of a container. The specified container can be a field on a page in the current book or any other container, such as a property or variable. If you clear the contents of a container on another page or on another background in a book, the current page does not change.

Using `clear` is not always the same as using `put null into` with the same string specifier. For example, if you use `clear` to clear the first line in a field, you remove the `crlf` as well as the text, and what was previously the second line becomes the first. If you use `put null into` to clear the first line in a field, the line retains its `crlf`, and all lines in the field retain their relative position.

Parameters

The `<object>` parameter is any expression that evaluates to a valid reference to an object.

The `<container>` parameter is any expression that evaluates to a valid container. Valid containers include properties, variables, and the Command window.

Examples

```
   -- Clears the contents of the sysError property
clear sysError

clear text of field id 4 of page 7
   -- Clears the contents of a field in
   -- the current book. The focus stays on the
   -- current page

clear textLine 3 of text of field id 2
   -- Clears line 3 of field id 2 on the current
   -- page, and line 4 becomes line 3

clear field "Data"
   -- Removes field Data and its contents

clear commandWindow
   -- Clears the contents of the Command window
```

closeFile Command

Syntax

closeFile <file name>

Description

Closes a file previously opened with the openFile or createFile command.

Parameter

The <file name> parameter is any expression that evaluates to a valid file name or a container that contains a valid file name, including the path name if necessary.

Example

```
-- Makes a copy of NEWDATA.TXT
set Original to "newdata.txt"
set Backup to "newdata.bak"
openFile Original
createFile Backup
readFile Original for 32767
while sysError <> "end of file"
   writeFile it to Backup
   readFile Original for 32767
end while
writeFile it to Backup
closeFile Original
closeFile Backup
```

closeRemote Command

Syntax

```
closeRemote [application <server name>] [topic <topic name>]
```

Description

The DDE command that closes the DDE channel between ToolBook and a server application. After ToolBook successfully executes a `getRemote`, `setRemote`, or `executeRemote` command, the DDE conversation that was opened for that command is left open until ToolBook returns to the top level. Open conversations are continued by subsequent DDE commands in which matching `<server name>` and `<topic name>` parameters are specified. A DDE conversation can be kept open after ToolBook returns to the top level by using the `keepRemote` command.

A DDE conversation can be explicitly closed at any time with the `closeRemote` command. A conversation kept open with `keepRemote` remains open until the instance is closed unless you explicitly close it with `closeRemote`.

Parameters

The `<server name>` parameter is any expression that evaluates to the DDE name of a Windows application. For instance, the DDE name of Microsoft Excel is `Excel`. If there is more than one instance of the application, only the first response is processed. If the `<server name>` parameter is omitted, any application that understands DDE protocol can respond.

The `<topic name>` parameter is any expression that evaluates to a topic recognized by the remote instance. The topic is usually a file name. See the DDE documentation for the appropriate application for the valid topics for that application. If there is more than one instance of the specified application with the specified topic, only the first response is processed. ToolBook recognizes `(Untitled)` or a file name as a valid topic. You can also omit the topic altogether or use the generic topic `system`, in which case any instance of the specified application can respond.

Example

```
closeRemote application "ToolBook" topic "Chart"
   -- Closes a DDE channel to a ToolBook application
   -- called "Chart"
```

continue Command

Syntax

```
continue
continue do
continue while
continue 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.

Parameter

The `<variable>` parameter in the `continue step` form of the command is the name of a variable in the enclosing `step` control structure. This variable indicates the `step` control structure on which `continue` is to operate (the control structure with that variable). If the `<variable>` parameter is omitted from the `continue step` form, the innermost `step` control structure is assumed. If a `<variable>` parameter is supplied, ToolBook executes the next iteration of the innermost enclosing step statement whose step variable is named by `<variable>`. For more details see the use of variables with the `step` control structure.

Examples

The following handlers generate numeric progressions.

```
to handle buttonUp
   local a,b
   put 0 into a
   put 0 into b
   while a < 10
      increment a
      put a after text of field "Value_1"
      if a mod 2 = 0
         continue while
            -- Proceed to next iteration if a is evenly
            -- divisible by 2, otherwise proceed
      end if
      increment b
      put b after text of field "Value_2"
   end while
end buttonUp

to handle buttonUp
   put 0 into a
   put 0 into b
   step i from 1 to 10
      increment a
      put a after text of field "A1"
      step j from 1 to a
         increment b
         put b after text of field "B1"
```

```
         if b > 5
            continue step i
               -- Go to the next iteration of the outermost
               -- step control structure
         end if
         increment a
         put a after text of field "A2"
      end step
   end step
end buttonUp
```

createFile Command

Syntax

`createFile <file name>`

Description

Creates a DOS file. 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, ToolBook sets the `sysError` property to `read only` and the `sysErrorNumber` property to 560. The contents of the file are not deleted and no file is created.

For details about writing to a file, see the `writeFile` command.

Parameter

The `<file name>` parameter is any expression that evaluates to a valid DOS file name, including the path name if necessary.

Example

```
-- The following creates a new file called newdata.dat
createFile "newdata.dat"
writeFile "Hello world" to "newdata.dat"
closeFile "newdata.dat"
```

deactivate menuItem

Command

Syntax

```
deactivate menuItem <menu item> [at <level>]
```

Description

Deactivates a user-defined menu command and dims the command name on a menu. A dimmed command cannot be chosen by a user. If a level is specified, the command is deactivated at that working level (`Reader`, `Author`, or `both`). If the level is not specified, the menu item deactivated at the working level in effect when the command is executed.

This command affects only user-defined commands, and does not affect the default ToolBook commands.

Parameters

The `<menu item>` parameter is the name of a command as it appears on a menu. If you want, you can omit leading numerals and any non-alphanumeric characters (such as spaces and punctuation, but not underscores) so that the name matches the message that ToolBook sends when a user chooses the menu item.

The `<level>` parameter must be an expression that evaluates to `author`, `reader`, or `both`.

Example

```
deactivate menuItem "Chapter One"
   -- Dims the Chapter One command at the
   -- current working level
```

decrement Command

Syntax

```
decrement <container> [by <amount>]
```

Description

Subtracts an amount from the value of an expression. This command is usually used in the context of a control structure. If the command does not specify an amount, the variable is decremented by 1.

If the value of the expression or amount is not numeric, an error occurs and ToolBook displays the Execution Suspended message box.

Parameters

The `<expression>` and `<amount>` parameters are expressions that yield a number.

Example

```
to get otherDistrictTotal
   local OtherDistricts,NumberOfDistricts
   set OtherDistricts to 135
   set NumberOfDistricts to the pageCount of this book
   step i from NumberOfDistricts to 0
      if text of field "ZipCode" contains 98001
         decrement OtherDistricts
            -- Subtracts 1 from OtherDistricts
      end if
      go to the next page
   end step
   return OtherDistricts
end otherDistrictTotal
```

draw Command

Syntax

```
draw <tool> from <location> to <location> [to <location>...]
```

Description

This command is used to create objects. `Draw` causes the same action that occurs when an author selects a tool, presses the mouse button at a location on the screen, and drags the mouse to another location on the screen before releasing the mouse button.

ToolBook puts the unique name of a newly drawn object into the `selection` property, which, among other things, allows you to set the name of the object so it can be referenced later.

Parameters

The `<tool>` parameter can be `field`, `button`, `recordField`, or any of the drawing tools:

angledLine	irregularPolygon	polygon
arc	line	rectangle
curve	pie	roundedRectangle
ellipse		

The `<location>` parameter specifies a point on the screen represented by two numbers separated by a comma. The first number is the distance from the left edge of the screen. The second number is the distance from the top of the screen. The `<location>` parameter can be any two expressions that are separated by a comma and that each yield a number, or any single expression that yields two numbers separated by a comma. The locations are interpreted in page units.

To draw an arc or pie, you must supply three points for each object. The first point is the starting point for the graphic. The second point is the sweep point, which indicates the direction of the arc or pie. The sweep point can be any of the points the mouse might move through if the arc or pie were being drawn with the mouse. The third point is the end point of the graphic.

To draw a curve, irregular polygon or angled line, you normally supply three or more points. Supplying only two points for an irregular polygon or angled line results in a straight line. For a curve, every other point starting with the first one is an anchor point. Every other point starting with the second, is a control point that determines the sharpness of the curve between the anchor points. When drawing with any of these tools, you can supply a maximum of 512 points.

Examples

```
draw rectangle from 1440,1440 to 6000,6000
   -- Draws a square from 1440,1440
   -- to 6000,6000 and sets its name so
   -- it can be referred to later in the script

set name of selection to "rect_1"
draw a line from 233,1500 to 233,2450
```

```
draw with line tool from 233,1500 to 233,2450
   -- Does the same thing as the previous example

draw an ellipse from 3440,1500 to 5440,7700

draw a field from 2006,2006 to 4000,4000

draw an arc from 2000,2500 to 2000,4000 to 3440,4000
   -- Draws an arc defined by the lower left quadrant of
   -- an ellipse

draw an irregularPolygon from 1000,1200 to 2000,1200 to \
   3505,1500 to 4000,4800 to 1000,1200

-- The following draws a checkerboard on a page and names
-- the squares A1 through H8
to handle buttonUp
   set sysStrokeColor to black
   set sysPattern to solidStroke
   set sysLineStyle to 1
   set sysFillColor to red
   step row from 1 to 8
      step column from 1 to 8
         draw rectangle from column*500, row*500 to \
            (column+1)*500, (row+1)*500
         set the name of the selection to\
            ansiToChar(row+64) & column
         if (row+column) mod 2 = 0 then
            set pattern of the selection to solidFill
               -- Makes a red square
         end if
      end step
   end step
end buttonUp
```

edit script

Command

Syntax

```
edit [the] script of <object>
```

Description

Opens the Script window and shows the script of a specified object. If this command is issued from a script, execution of the script is suspended until the Script window is closed. This command can be used to allow a reader to edit a script in the Script window.

Scripts that include the `edit script` command will cause an error in Runtime ToolBook.

Parameter

The `<object>` parameter is an expression that evaluates to the name or ID of an object.

Example

```
edit script of button "More"
   -- Displays the Script window with the script of
   -- button More

edit script of this background
   -- Edits script of current background
```

execute

Command

Syntax

```
execute <source>
```

Description

Gets whatever is in `<source>` and executes it as one or more OpenScript statements. The statements are executed in the context of the current handler.

Parameter

The `<source>` parameter is an expression that evaluates to one or more OpenScript statements or to any container that contains OpenScript statements.

Example

```
execute text of field "MyScript"
   -- If field MyScript contains OpenScript
   -- statements, those statements are executed
```

executeRemote Command

Syntax

```
executeRemote <command> [application <server name>] [topic
   <topic>]
```

Description

Sends a command to the current page in another instance of ToolBook or to another Windows application, using Windows Dynamic Data Exchange (DDE) protocol. For details about DDE, see "Using Windows Dynamic Data Exchange" in Chapter 6, "Beyond the Basics," of Using OpenScript.

Immediately after ToolBook executes an `executeRemote` command, the value of the `sysError` property is set to a comma-separated list of nine items. The first item indicates the status of the remote request. The last eight items describe the LOBYTE of the WM_DDE_ACK DDE message, which does not have any meaning to ToolBook but may have meaning to the application you are exchanging data with. For details about the meaning of these items, see the DDE documentation for the appropriate application. The `sysErrorNumber` and `sysError` properties are set to a value corresponding to the status of the remote request as described in the following table.

SysErrorNumber and sysError values for the executeRemote command

`sysErrorNumber`/ `sysError`	Meaning
8123 `OK.`	The application responded to the request
8124 `Failed: Denied.`	The application responded but could not or would not satisfy the request
8125 `Failed: Busy.`	The application responded but is doing something else and failed to perform the requested action
8126 `Failed: Memory Error.`	ToolBook did not have enough global or local memory to generate the request or accept a response
8127 `Failed: No Server.`	No applications responded to the request
8128 `Failed: Interrupted.`	The application responded, but the connection was broken before the application acknowledged the command

The `executeRemote` command does not bypass normal password protections.

Parameters

The `<command>` parameter is any expression that evaluates to one or more

OpenScript statements or to a valid command string for an application. If the commands are sent to another application, see the documentation for that application for the format of the command string.

The `<application>` parameter is any expression that evaluates to the DDE name of a Windows application. For instance, the DDE name of Microsoft Excel is `Excel`. If there is more than one instance of the application, only the first response is processed. If the `<application>` parameter is omitted, any application that understands DDE protocol can respond.

The `<topic>` parameter is any expression that evaluates to a topic recognized by the remote instance. The topic is usually a file name. See the DDE documentation for the appropriate application for the valid topics for that application. If there is more than one instance of the specified application with the specified topic, only the first response is processed. ToolBook recognizes `(Untitled)` or a file name as a valid topic. You can also omit the topic altogether or use the generic topic `system`, in which case any instance of the specified application can respond.

Examples

```
executeRemote "text of field id 1" \
   application ToolBook topic "Workbook.tbk"
      -- If field id 1 contains OpenScript statements,
      -- those statements are executed

executeRemote "send reader; go to page 5 of book" \
   && quote & "Book2.tbk" & quote application ToolBook

executeRemote "[select(""r1c1:r3c3"")][new(2)]" \
   application Excel topic "Sheet1"
   -- Excel selects the range and creates a bar chart for it
```

export Command

Syntax

```
export <file name> as <type> using <fields>
```

Description

Exports the text from all record fields of the current background of the current book into an ASCII text file, inserting quotation marks to surround the exported contents of each record field and inserting `crlf` to delimit each record. If the file does not exist, ToolBook creates it. If the file exists and is a read-only file, an error occurs and ToolBook displays the Execution Suspended message box. If the file exists and ToolBook can write to it, any existing contents of the file are lost. For details about exporting files, see Chapter 10, "Managing ToolBook Files," in Using ToolBook.

If you try to export a file when there are 10 files already open, an error occurs and ToolBook displays the Execution Suspended message box.

Parameters

The `<file name>` parameter is any expression that evaluates to a valid file name, including the path name if necessary.

The `<type>` parameter is `fixed` or `delimited`. If `<type>` is `fixed`, the `<fields>` parameter must be an expression that evaluates to a comma-separated list of integers indicating the field lengths and justification. Field length specifiers can be positive or negative. If a field length is positive, text in the field is exported left-justified and any spaces needed to fill the field are inserted on the right. If the field length is negative, the text in the field is exported right-justified and any spaces needed to fill the field are inserted on the left. If the length of text in a field is larger than the specified size, the field is truncated on the right.

If `<type>` is `delimited`, then `<fields>` must be an expression that evaluates to the field separator. The field separator can be a single character, such as a semicolon, or a constant that represents a single character, such as `space`. However, the field separator cannot be a quotation mark (`"`). To use tab characters as the field separator, specify the constant, `Tab`, as the `<fields>` parameter.

Example

```
export "mybook.dat" as delimited using ";"
```

extend select

Command

Syntax

```
extend select <objects>
```

Description

Extends the current selection to include specified objects. You can select only the current page or objects on the current page or background.

Also see the `select` command.

Parameter

The `<objects>` parameter is an expression that evaluates to a comma-separated list of one or more object identifiers. The following types of objects can be selected:

angledLine	group	pie
arc	irregularPolygon	polygon
button	line	recordField
curve	page	rectangle
ellipse	paintObject	roundedRectangle
field	picture	

Examples

```
-- Adds the object clicked to the current selection
-- if the shift key is pressed during the click. Otherwise,
-- makes the clicked object the selection.
to handle buttonUp location,is_shift,is_ctrl
   if is_shift is true and the selection is not null
      extend select target
   else
      select target
   end if
end buttonUp
```

flip 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 pages from the current page to the last page, flipping continues from the beginning of the book. The `flip` form flips one page. The `flip all` form flips all pages, starting and ending at the current page.

Parameter

The `<number>` parameter is any expression that yields a positive integer.

Examples

```
to handle buttonUp
   flip all
      -- Flips through all pages in a book
end buttonUp

to handle buttonUp
   flip 5
end buttonUp

to handle buttonUp
   flip 5 pages
end buttonUp

-- The following handler flips all the pages in a book
-- without displaying them. This "caches" the pages in
-- memory if there is room and makes subsequent page
-- flipping faster (this technique is useful for page
-- animation).
to handle buttonUp
   set sysLockScreen to true
   flip all
end
```

format Command

Syntax

format [<type>] <container> [as <new format>] [from <old format>]

Description

Converts the contents of a container to a specified format.

If you specify both <old format> and <new format>, the container's value is converted from the format specified by <old format> to the format specified by <new format>.

If <new format> is specified but <old format> is not, the value in the container is assumed to be formatted according to the current value of the appropriate system property (sysDateFormat, sysNumberFormat, or sysTimeFormat).

If <old format> is specified but <new format> is not, the value in the container is assumed to be formatted as specified by <old format> and are converted to the format specified by the appropriate sysFormat property.

If both <new format> and <old format> are omitted, the value in the container is assumed to be formatted according to the current value of the appropriate sysformat property and is converted to the default format of null.

For details about how ToolBook reads numeric values, see "Values and Expressions of Values" in Chapter 2, "Script Basics", of Using OpenScript.

Parameters

The <type> parameter is number, date, or time. The default value is number. The <container> parameter is an expression that evaluates to the name of a container that contains the value you want to format. If the <type> parameter is number, the contents of the container must be numeric. The <new format> and <old format> parameters are valid numeric, date, or time formats as described in the following tables.

Numeric formats

You can define the number format you want by building a string containing any of the following symbols. The string must be enclosed in quotes.

Number format symbols

Symbol	Meaning
null	Default format. ToolBook displays the number as precisely as possible, using a period as the decimal separator, and using scientific notation when necessary.
?	General precision format. If used as <new format>, ToolBook displays the number as precisely as possible using the value of sysDecimal as the decimal separator. If used as <old format>, ToolBook will discard characters matching sysCurrency and sysThousand, and replace the sysDecimal character with a period before interpreting the

	number.
`0`	Digit placeholder. If the number has fewer digits on either side of the decimal point than there are zeros on either side of the decimal point, ToolBook displays the extra zeros. If the number has more digits to the right of the decimal point than there are zeros to the right of the decimal point in the format, ToolBook rounds the number to as many decimal places as there are zeros to the right of the decimal point in the format. If the number has more digits to the left of the decimal point than there are zeros to the left of the decimal point in the format, ToolBook displays the extra digits.
`#`	Digit placeholder. Similar to 0 above, except that ToolBook uses space padding instead of extra zeros if the number has fewer digits on either side of the decimal point than there are `#`s on either side of the decimal point in the format.
.	Decimal point. If the format contains only `#` characters to the left of this symbol, ToolBook begins numbers smaller than 1 with a decimal point. To avoid this, use 0 as the first digit placeholder to the left of a decimal point.
,	Thousands separator. ToolBook separates thousands by commas if the format contains a comma surrounded by `#`s or `0`s.
`E- E+ e- e+`	Scientific notation. If a format contains one `0` or `#` to the right of an `E-`, `E+`, `e-`, or `e+`, ToolBook displays the number in scientific notation and inserts an `E` or `e`. The number of `0`s or `#`s to the right determines the number of digits in the exponent. Use `E-` or `e-` to place a minus sign by negative exponents. Use `E+` or `e+` to place a minus sign by negative exponents and a plus sign by positive exponents.
Any other character	Literal text. Any printable character besides the above symbols is displayed literally. For example, if you want a dollar sign to precede all numbers, you can specify a format like `"$###.00"`. Embedding literal text within a number format may give unexpected results. For example, if a word containing a `d` is used in a number format, the `d` is interpreted as a date format as described in the following table.

Date formats

A date format is a string of characters and numbers. The string can be any text and any combination of the formats shown in the following table. The default value is `m/d/yy`.

Date Format Symbols

Format	Description
`M`	The month's complete name.
`MMM`	The month's name abbreviation.

`m`	The month's number (1, 2, ...12)
`mm`	The month's number with a leading 0 for 1 through 9 (01, 02, ...12)
`d`	The day's number (1, 2, ...31)
`dd`	The day's number with a leading 0 for 1 through 9 (01, 02, ...31)
`y`	The year's number as an integer
`yy`	The last two digits of the year's number
`yyyy`	The year's number as four digits
`h`	The hour's number on a 12-hour clock (1, 2, ...12)
`hh`	The hour's number on a 12-hour clock with a leading 0 for 1 through 9 (01, 02, ...12)
`h24`	The hour's number on a 24-hour clock (0, 1, ...23)
`hh24`	The hour's number on a 24-hour clock with a leading 0 for 0 through 9 (00, 01, ...23)
`min`	The minute's number (00, 01, ...59)
`sec`	The second's number (00, 01, ...59)
`AMPM`	The local language equivalent for `AM` if the date is in the morning and `PM` if the date is in the afternoon as defined by S1159 and S2359 settings in WIN.INI.
`seconds`	The number of seconds elapsed since 00:00:00 GMT on January 1, 1970

Time formats

A time format is a string of characters and numbers. The string can be any combination of the formats shown in the following table. The default format is `h:min:sec`.

Time Format Symbols

Format	Description
`h`	The hour's number on a 12-hour clock (1, 2, ...12)
`hh`	The hour's number on a 12-hour clock with a leading 0 for 1 through 9 (01, 02, ...12)
`h24`	The hour's number on a 24-hour clock (0, 1, ...23)
`hh24`	The hour's number on a 24-hour clock with a leading 0 for 0 through 9 (00, 01, ...23)
`min`	The minute's number (00, 01, ...59)

`sec`	The second's number (00, 01, ...59)
`AMPM`	The local language equivalent for `AM` if the date is in the morning and `PM` if the date is in the afternoon as defined by S1159 and S2359 settings in WIN.INI.
`seconds`	The number of seconds elapsed since 00:00:00 GMT on January 1, 1970

Examples

```
format date textline 1 of text of field "Date" as\
   "y MMM dd" from "mm/dd/yy"
   -- Formats line 1 of field Date as a date

format text of field "Quantity" as "#.00"
      -- Formats the contents of field
      -- Quantity as a number

format time last textline of footer from "hh24:min:sec"
      -- Formats the last line of the
      -- footer as sysTimeFormat
```

forward Command

Syntax

```
forward
forward <message> [<parameters>]
forward to system
```

Description

Passes a message up the object hierarchy. If no parameters are provided, the message that is currently being handled is forwarded to the next object up the hierarchy, with the current handler parameters as the parameters of the message. If the `forward to system` form of the command is used, the message that is currently being handled is forwarded directly to ToolBook (the system), with the current handler parameters as the parameters of the message. After the message is processed, script execution continues at the statement following the `forward` command.

The `forward` command can be used to send a message up the hierarchy and override the default behavior for a message, as shown in the following examples. The `forward` command is different from the `send` command in that the `send` command is usually used to make a procedural call to perform a task for the handler sending the message.

Also see the `send` command.

Parameters

The `<message>` parameter is an expression that evaluates to any valid built-in or user-defined message. Valid messages are strings that begin with a letter and contain no spaces or punctuation marks except the underscore character. The optional `<parameters>` parameter is one or more expressions, separated by commas if there is more than one.

Example

```
to handle buttonDown location
   system clickSpot
   push location onto clickSpot
   forward
      -- Forward the message in case it also needs to be
      -- handled at another level in the hierarchy
end buttonDown
```

fxDissolve

Command

Syntax

```
fxDissolve [<speed>] [to <result>]
```

Description

A visual effect that causes the current page to dissolve to black, gray, white, or another page. The dissolve can be slow, medium, or fast.

Parameters

The `<speed>` parameter must be `slow`, `normal`, or `fast`; the default value is `normal`. The `<result>` parameter is an expression that evaluates to a page in a book or the word `black`, `gray`, or `white`; the default value is `black`.

Examples

```
-- When the Next command is chosen from the menu,
-- do a fast dissolve to the next page
to handle Next
   fxDissolve fast to next page
end Next

-- When the Next command is chosen from the menu,
-- do a slow dissolve to white then go to then next page
to handle Next
   fxDissolve slow to white
   go to next page
end Next
```

fxWipe

Command

Syntax

```
fxWipe <direction> [<speed>] [to <result>]
```

Description

A visual effect that causes the current page to appear as if it is being wiped off the screen by another page or by solid black, gray, or white. The wipe can be slow, medium, or fast. If a result is not specified, the page is wiped to black.

Parameters

The `<direction>` parameter must be `top`, `bottom`, `left`, or `right`, indicating which edge of the screen the wipe will move toward. The `<speed>` parameter must be `slow`, `normal`, or `fast`; the default value is `normal`. The `<result>` parameter is an expression that evaluates to a page identifier or the word `black`, `gray`, or `white`; the default value is `black`.

Example

```
-- When the Next command is chosen from the menu, wipe the
-- current page off to the left leaving black. Then, wipe
-- the next page onto the screen from right to left.
to handle Next
   fxWipe left fast
   fxWipe left fast to next page
end Next
```

fxZoom

Command

Syntax

```
fxZoom [<speed>] [to <result>] [at <location>]
```

Description

Causes the visual effect of a circular wipe from the center of a page or from a specified point on a page to another page or a screen. The zoom can be slow, medium, or fast. If a result is specified, the screen is zoomed to black, gray, white, or a specified page.

Parameters

The `<speed>` parameter must be `slow`, `normal`, or `fast`; the default value is `normal`.

The `<result>` parameter is an expression that evaluates to a page identifier or the word `black`, `gray`, or `white`; the default value is `black`.

The `<location>` parameter specifies a point on the screen, represented by two numbers separated by a comma. The first number is the distance from the left edge of the window (measured in page units). The second number is the distance from the top of the window (measured in page units). The `<location>` parameter can be any two expressions that are separated by a comma and each yield a number, or any single expression that yields two numbers separated by a comma.

Example

```
-- When the Next command is selected from the menu, zoom
-- to the next page from location 2035, 1500
to handle Next
   put "2034,1500" into It
   fxZoom slow to next page at it
end Next
```

get Command

Syntax

```
get <expression>
```

Description

Evaluates an expression and places the result in the local variable `It`. For details about the variable `It`, see "Variables" in Chapter 2, "Script Basics," of Using OpenScript.

Parameter

The `<expression>` parameter is any expression that yields a value.

Examples

```
get my bounds
   -- Gets value of the bounds property for object whose
   -- script is currently executing and puts the value into
   -- It

get the sysDate
   -- Gets the system date and puts the value into It
```

getRemote Command

Syntax

```
getRemote <data> [application <server name>] [topic <topic>]
```

Description

Gets the value of any property or expression in another instance of ToolBook or in another Windows application, using Windows Dynamic Data Exchange (DDE) protocol. The data that is returned is put into `It`. For details about DDE, see "Using Windows Dynamic Data Exchange" in Chapter 6, "Beyond the Basics," of Using OpenScript.

After ToolBook successfully executes a `getRemote` command, the DDE conversation that was opened for that command is left open until ToolBook returns to the top level. Open conversations are continued by subsequent DDE commands in which matching <server name> and <topic name> parameters are specified. A DDE conversation can be kept open after ToolBook returns to the top level by using the `keepRemote` command.

Immediately after ToolBook executes the `getRemote` command, the value of the `sysError` property is set to a list of nine items. The first item indicates the status of the remote request. The last eight items describe the `LOBYTE` of the WM_DDE_ACK DDE message, which does not have any meaning to ToolBook but may have meaning to the application you are exchanging data with. For details about the meaning of these items, see the DDE documentation for the appropriate application. The `sysErrorNumber` property is set to a number corresponding to the status of the remote request as shown in the following table.

SysErrorNumber values for the getRemote command

`sysErrorNumber`/ `sysError`	Meaning
8123 `OK`.	The application responded to the request
8124 `Failed: Denied`.	The application responded but could not or would not satisfy the request
8125 `Failed: Busy`.	The application responded but is doing something else and failed to perform the requested action
8126 `Failed: Memory Error`.	ToolBook did not have enough global or local memory to generate the request or accept a response
8127 `Failed: No Server`.	No applications responded to the request
8128 `Failed: Interrupted`.	The application responded, but the connection was broken before the application acknowledged the command

The `getRemote` command does not bypass normal password protections.

Parameters

The `<data>` parameter is any expression that yields a value that identifies the data you want to get. If you're getting data from another instance of ToolBook, the expression can yield the name of any valid container in that instance.

The `<application>` parameter is an expression that evaluates to the DDE name of a Windows application. For instance, the DDE name of Microsoft Excel is `Excel`. If there is more than one instance of the application, only the first response is processed. If the `<application>` parameter is omitted, any application that understands DDE protocol can respond.

The `<topic>` parameter is any expression that evaluates to a topic recognized by the remote instance. The topic is usually a file name. See the DDE documentation for the appropriate application for the valid topics for that application. If there is more than one instance of the specified application with the specified topic, only the first response is processed. ToolBook recognizes `(Untitled)` as a valid topic if the currently displayed book is untitled. Otherwise, ToolBook recognizes the file name of the currently open book. You can also omit the topic altogether or use the generic topic `system`, in which case any instance of a specified application can respond.

Examples

```
getRemote "target" application ToolBook
   -- Gets the value of the target property from
   -- the next instance of ToolBook and puts
   -- the value into It

getRemote "pageCount of this book" application ToolBook \
   topic "Mariposa.tbk"
   -- Gets the number of pages in the book
   -- "Mariposa.tbk" that is running in
   -- another instance of ToolBook, and puts
   -- the value into It

getRemote "r1c1:r3c3" application Excel topic "Revenue.xls"
   -- Gets the values of the nine cells of the
   -- range r1c1:r3c3 and puts them into It
```

go Command

Syntax

```
go [to] <expression>
```

Description

Goes to any page in the currently open book, or any page in any other book. If the destination is a book and a page is not specified, ToolBook goes to the first page in the book.

If the statement `go next page` is encountered on the last page of a book, ToolBook goes to the first page in the book. If the statement `go previous page` is encountered on the first page of a book, ToolBook goes to the last page of the book.

Parameter

The `<expression>` parameter is any expression that evaluates to a valid identifier for a page or book. For details about valid page and book identifiers, see "Referring to Objects" in Chapter 2, "Script Basics," of Using OpenScript.

Examples

```
go to book "c:\mybooks\contacts.tbk"
   -- Goes to the first page of the book, contacts.tbk

go to page 5 of book "lessons.tbk"

go to page 10 of this book

go next page
```

hide

Command

Syntax

```
hide [<object>]
```

Description

Removes a specified object from view. The result of this command is the same as setting the `visible` property for the object to `false`. Hidden objects do not receive mouse or keyboard event messages, and hidden fields and buttons are not included in the tabbing order.

Also see the `show` command.

Parameter

The `<object>` parameter can be any expression that evaluates to an object, palette, or window that has a `visible` property.

Hideable objects are:

angledLine	irregularPolygon	polygon
arc	line	polygonPalette
button	linePalette	recordField
colorTray	mainWindow	rectangle
commandWindow	menuBar	roundedRectangle
curve	paintObject	scrollBar
ellipse	patternPalette	spectrumPalette
field	picture	statusBox
group	pie	toolPalette

If no `<object>` parameter is given, the current selection is hidden.

Examples

```
hide the commandWindow
   -- Hides the Command window

hide field id 1 of this background
   -- Hides field id 1 on the current background

hide recordField "Security level"
   -- Hides the record field called "Security level"
```

import Command

Syntax

```
import <file name> as <type> using <fields>
```

Description

Imports a text file into the current book. The file can have fixed-length or delimited text fields. If the file does not exist, ToolBook sets the `sysError` propterty to `no such file` and the `sysErrorNumber` property to 559. For details about importing files, see Chapter 10, "Managing ToolBook Files," in Using ToolBook.

If you try to import a file and there are 10 files already open, an error occurs and ToolBook displays the Execution Suspended message box.

You can import quoted regions of delimited files. Quoted regions are text strings enclosed in quotation marks. All field delimiters and record delimiters are ignored in quoted regions. To preserve quotation marks within a quoted region, a second quotation mark must immediately follow the quotation mark to be preserved. For example, the field must contain `"this is a ""quoted"" region"` to get `this is a "quoted" region`.

Parameters

The `<file name>` parameter is any expression that evaluates to a valid file name, including the path name if necessary. The `<type>` parameter is `fixed` or `delimited`. If `<type>` is `fixed`, the `<fields>` parameter must be an expression that evaluates to a comma-separated list of positive integers indicating the field lengths. If a field is larger than the specified size, the field is truncated on the right. If `<type>` is `delimited`, the `<fields>` parameter must be an expression that evaluates to the field separator.

The field separator can be a single character, such as a semicolon, or a constant that represents a single character, such as `space`. However, the field separator cannot be a quotation mark (`"`). To use tab characters as the field separator, use the constant, `Tab`, as the `<fields>` parameter.

Examples

```
import "maillist.dbs" as delimited using ","
   -- Imports a delimited-field file that
   -- has comma field separators

import "titles.dbs" as delimited using space
   -- Imports a delimited-field file that
   -- has a space character as the field separator

import "cashflow.xls" as fixed using "10,5,7,9,14"
   -- Imports a fixed-field file
   -- that has field lengths of 10, 5, 7, 9, and 14
```

importGraphic

Command

Syntax

```
importGraphic <file name>
```

Description

Used to import a specifed file as a picture object or paint object.

Parameters

The `<file name>` parameter is any expression that evaluates to a valid graphics file name, including the path name if necessary. For file formats other than .BMP, .DIB, or.WMF, an import filter for that file format must be installed in the user's path; otherwise, ToolBook cannot import the file. If the imported file has the extension .BMP or .DIB, ToolBook creates a paint object. For other file name extensions, ToolBook creates a picture.

Examples

```
importGraphic "ram.dib"
   -- imports a ".dib" file called "ram.dib" as a paint
   -- object
```

increment

Command

Syntax

```
increment <container> [by <amount>]
```

Description

Adds an amount to the value of an expression. The `increment` command is normally used in the context of a control structure.

The values of the expression and the amount must be numeric, or an error occurs and ToolBook displays the Execution Suspended message box.

Parameters

The `<expression>` and `<amount>` parameters are any expressions that yield a number. The default value for the `<amount>` parameter is `1`.

Example

```
conditions
when ZipCode is 98001
   increment NorthDistrict
when ZipCode is 98002
   increment EastDistrict
when ZipCode is 98003
   increment WestDistrict
when ZipCode is 98004
   increment SouthDistrict
else
   increment Uncategorized
end conditions
```

keepRemote Command

Syntax

```
keepRemote [application <server name>] [topic <topic name>]
```

Description

The DDE command that keeps the DDE channel between ToolBook and a server application open. After ToolBook successfully executes a `getRemote`, `setRemote`, or `executeRemote` command, the DDE conversation that was opened for that command is left open until ToolBook returns to the top level. Open conversations are continued by subsequent DDE commands in which matching `<server` name> and `<topic name>` parameters are specified. A DDE conversation can be kept open after ToolBook returns to the top level by using the `keepRemote` command.

A DDE conversation can be explicitly closed at any time with the `closeRemote` command. A conversation kept open with `keepRemote` remains open until the instance is closed unless you explicitly close it with `closeRemote`.

Parameters

The `<server name>` parameter is any expression that evaluates to the DDE name of a Windows application. For instance, the DDE name of Microsoft Excel is `Excel`. If there is more than one instance of the application, only the first response is processed. If the `<server name>` parameter is omitted, any application that understands DDE protocol can respond.

The `<topic name>` parameter is any expression that evaluates to a topic recognized by the remote instance. The topic is usually a file name. See the DDE documentation for the appropriate application for the valid topics for that application. If there is more than one instance of the specified application with the specified topic, only the first response is processed. ToolBook recognizes `(Untitled)` or a file name as a valid topic. You can also omit the topic altogether or use the generic topic `system`, in which case any instance of the specified application can respond.

Example

```
keepRemote application "ToolBook" topic "Chart"
   -- Opens a DDE channel to a ToolBook application
   -- called "Chart"
```

local

Command

Syntax

```
local [variable[s]] <variable list>
```

Description

Makes a variable name known and its contents available within the context of the current handler. The value of a local variable is known to ToolBook only within the handler in which the local variable is defined. ToolBook considers all undeclared variables to be local variables. The value of a local variable persists only during execution of the handler in which it is defined. ToolBook initializes the value of a local variable to null when the local variable is declared.

For details about variables, see "Variables" in Chapter 2, "Script Basics," of Using OpenScript. Also see the `system` command.

Parameter

The `<variable list>` parameter contains one or more variable names, separated by commas if there is more than one.

Example

```
-- This handler counts the number of objects
-- in a book and puts the number into the Command window.
-- The variables, objCnt and i are declared as local
-- variables.
to handle buttonUp
   local objCnt, i
   set objCnt to 0
   set i to 0
   step i from 1 to pageCount of this book
      set objCnt to objCnt + itemCount(objects of page i)
   end step
   put objCnt into the commandWindow
end buttonUp
```

magnify Command

Syntax

```
magnify [<power>] [at <location>]
```

Description

Magnifies the view of the current page. The `<location>` parameter defines the center of the area to be magnified. This command is equivalent to clicking the `Zoom` tool.

Parameters

The `<power>` parameter must be `1`, `2`, `4`, `8`, or `16`, and indicates the magnification level. If the `<power>` parameter is omitted, the default value is `2`.

The `<location>` parameter specifies a point on the screen represented by two numbers separated by a comma. The first number is the distance from the left edge of the screen. The second number is the distance from the top of the screen. The `<location>` parameter can be any two expressions separated by a comma which each yield a number, or any single expression that yields two numbers separated by a comma. The locations are interpreted in page units. If the `<location>` parameter is omitted, the default is the center of the current page.

Examples

```
magnify 4
   -- Magnifies the current page 4x at the center of the
   -- page

magnify 1
   -- Returns the page to normal view

magnify 2 at 2400,3650
   -- Magnifies an area whose center is at 2400,3650
```

move Command

Syntax

```
move [<object>] by <amount>
move [<object>] to <location>
```

Description

Moves one or more objects, a window, the selection, or a multiple selection either to a location or by a specified amount.

Parameters

The `<object>` parameter is an expression that evaluates to the name or ID of an object on a page or background, or to a comma-separated list of object names or IDs. The default is the current selection.

The `<amount>` parameter specifies an offset from the object's current position. The offset is represented by two numbers separated by a comma. The first number represents the horizontal offset from left to right, and the second number represents the vertical offset from top to bottom. If the specified object is a window or palette, both offsets are measured in screen units (pixels). Otherwise, both offsets are measured in page units.

The `<location>` parameter specifies a point on the screen. The point is represented by two numbers separated by a comma. If the specified object is a window or palette, the first number represents the distance from the left edge of the screen, and the second number represents the distance from the top edge of the screen, both distances measured in screen units (pixels). Otherwise, the first number represents the distance from the left edge of the page, and the second number represents the distance from the top of the page, both distances measured in page units.

The `<amount>` or `<location>` parameter is any two expressions that are separated by commas and each yield a number, or any single expression that yields two numbers separated by a comma.

Examples

```
move the commandWindow to 1,1
   -- Moves the Command window to the top left corner of
   -- the screen

move button "Next" by 1440,-1440
   -- Moves button Next down and to the left one inch
```

openFile Command

Syntax

`openFile <file name>`

Description

Opens a DOS file. You can open a file to read in text using the `readFile` command or to append information to the file using the `writeFile` command. If the specified file doesn't exist, ToolBook sets the `sysError` property to `no such file` and the `sysErrorNumber` property to 559.

If you try to open a file and there are already 10 files open, an error occurs and ToolBook displays the Execution Suspended message box. The same error occurs if there are 10 files open and you try to import or export a file.

Parameter

The `<file name>` parameter is any expression that evaluates to a valid file name, including the path name if necessary.

Example

```
if text of field "Choice" = "Yes" then
   openFile "a:\romans.dat"
   if sysErrorNumber is 559 then
      request "Please insert the disk containing the file"
         -- Displays a message if the file
         -- romans.dat does not exist
   end if
else
   go to last page
end if
```

pause Command

Syntax

```
pause <time> [ticks]
pause <time> seconds
```

Description

Causes ToolBook to wait for a specified length of time before executing the rest of the script. If `seconds` is not specified, ticks are used. A tick is approximately 1/100th of a second.

Parameter

The `<time>` parameter is any expression that yields a number. The number may be in the range 0 through 2^31.

Example

```
-- The following draws a triangle, pausing after each
-- line segment
draw with line from 1025,1025 to 2422,2021
pause 2 seconds
draw with line from 2422,2021 to 1025,2021
pause 2 seconds
draw with line from 1025,2021 to 1025,1025
```

pop Command

Syntax

```
pop [<stack>] [into | before | after] [<destination>]
```

Description

Retrieves the first item of a stack. A stack is any list of comma-separated items.

If a stack is not specified, the item is popped from the `sysHistory` property. An item popped from `sysHistory` consists of the full identifier for a page. If no destination is given, the popped item goes into `It`. After the item has been popped, the item, and the comma following it if there is more than one item in the stack, is removed from the stack and the number of items in the stack is reduced by one. If you attempt to pop an item from an empty stack, an error occurs and ToolBook displays the Execution Suspended message box.

The `pop...into` form of the command replaces the contents of the destination with the value of the popped item. The `pop...before` form of the command inserts the value of the popped item before the contents of the destination. The `pop...after` form of the command appends the value of the popped item after the contents of the destination.

Also see the `push` command and the `sysHistory` property.

Parameters

The `<stack>` parameter is any expression that evaluates to a comma-separated list of items. The `<destination>` parameter is any expression that evaluates to the name of a container.

Examples

```
pop into text of field 1
   -- Moves the uniqueName of the first page from
   -- sysHistory into field 1 of the current page

pop sysHistory into it
go to it
   -- Goes to the first page in the system history

to get onPage obj
   -- Checks if an object is within boundaries of a page
   local verts, height, width
   put item 1 of the size of this book into width
   put item 2 of the size of this book into height
   put vertices of obj into verts
   while verts <> null
      pop verts      -- Pop the X coordinates into It
      if (it < 0) or (it > width) then
         return false
      end if
      pop verts   -- Pop the Y coordinates into It
      if (it < 0) or (it > height) then
         return false
```

```
      end if
   end while
   return true
end onPage
```

print Command

Syntax

```
print [<number> [pages]]
```

Description

Prints a book starting at the current page. You can print all the pages in a book or a selection of pages.

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 some pages in the range do not satisfy, ToolBook will print fewer than the specified number of pages.

If the number of pages specified is larger than the number of pages from the current page to the last page, ToolBook stops printing after printing the last page of the book. If all the pages are to be printed, printing begins with the current page through the last page and then continues from the first page to the current page.

This command can be used only within a `start spooler` control structure. If you want printing to begin from a particular page of a book, specify the starting page for printing using a `go` command within the `start spooler` control structure.

You can set printer properties to customize printing. For details about the different printer properties, see the entries for:

printerArrangement		
	printerFieldWidths	printerLeftMargin
printerBorders	printerGroupsAcross	printerMargins
printerBottomMargin	printerGutterHeight	printerName
printerClipText	printerGutters	printerRightMargin
printerConditions	printerGutterWidth	printerStyle
printerFieldNames	printerLabelWidth	printerTopMargin
printerFields		

To print a report, you must set the `printerStyle` property to `columns` or `groups`.

Parameter

The `<number>` parameter is the word `all` or a non-negative integer indicating the number of pages to be printed, beginning with the current page. If you want to make sure that you print all of the pages from the current page to the end of the book, you can use a large number for the `<number>` parameter.

Examples

```
-- The following prints all pages where there is text in
-- field Name
set printerConditions to "text of field" &&\
    quote & "Name" & quote && "of this page is not null"
start spooler
   go to first page
   print all pages
```

```
end spooler

-- The following prints a group report consisting of 15
-- pages in the current background, including only the
-- pages where 9 is the first character in field id 5,
-- perhaps to print only addresses with zip codes beginning
-- with a 9
set printerStyle to "groups"
set printerConditions to \
   "char 1 of word 1 of text of field id 5 = 9"

start spooler
   print 15
end spooler
```

print eject

Command

Syntax

```
print eject
```

Description

Advances the printer to the top of the next sheet. This command can be used only within a `start spooler` control structure.

Example

```
-- Use the print eject command to eject a blank page
-- after each page printed
start spooler
   print 1 pages
   print eject
   print 1 pages
   print eject
end spooler
```

push Command

Syntax

```
push [<expression>] [onto <stack>]
```

Description

Adds an item to a stack. A stack is any list of comma-separated items. The items can be any value. If a stack is not specified and the item is the unique name of a page, the item is pushed onto the `sysHistory` property. If the item is not the uniqueName of a page, a stack must be specified.

The `push` command adds the item to the beginning of the list. In other words, the item is pushed onto the top of the stack. If you prefer, you can push an item elsewhere on the stack, as illustrated in the last example in this entry.

Items pushed onto a stack can be retrieved using the `pop` command. The `push` and `pop` commands are useful, for example, when you have a summary page in a book that you want the reader to return to at the end of each of several sections. Instead of using the command `go to page 1`, you can use the statement `push page "Summary" onto bookmarks` (where `bookmarks` is the name of a system variable used as a stack). When it's time for the reader to return to the summary, you can use the statement `pop bookmarks` to pop the page into `It` and then use a `go to it` statement to display the page with the summary. That way, the `Summary` page will still be displayed even if its page number has changed.

You can use a stack like a one-dimensional array and access items in the stack using string specifier expressions. Note that pushing and popping numbers is much faster than building or accessing a stack using the `put` and `get` commands.

For more details see the `pop` entry.

Parameters

The `<expression>` parameter is any expression that yields a value.

The `<stack>` parameter is an expression that evaluates to anything that can contain a list, such as a variable or a text line in the text of a field. The default value is `sysHistory`. Note that if you try to push an item other than the unique name of a page onto `sysHistory`, an error occurs and ToolBook displays the Execution Suspended message box. To see what items are in the system history stack, look at the value of the `sysHistory` property. It contains a list of the uniqueNames of the items in the system history.

To avoid unpredictable results when pushing items onto a variable, declare the variable before pushing items onto it.

Examples

```
-- This handler is for the script of a button on page
-- "Summary" to go to the next page
to handle buttonUp
   system Bookmarks
   push page "Summary" onto Bookmarks
      -- Push the uniqueName of the Summary page onto
      -- a stack
```

```
   go to next page
end buttonUp

-- You could put this handler in a page script to respond
-- to a user-defined menu command called Go To Summary
to handle goToSummary   -- Handles a user-defined message
   system Bookmarks
   pop bookmarks
      -- Put the last item pushed into Bookmarks onto It
   go to it
      -- Go to the page pushed onto It
end goToSummary

-- This user-defined function reverses the order of items
-- in a stack
to get reverseList stack
   local tempStack
   put null into tempStack
   while stack <> null
      pop stack
         -- Pops the first item in the stack into It
      push it onto tempStack
   end while
   return tempStack
end reverseList

-- This example shows how you can push an item to a specific
-- place in a stack
push "cat" onto animals
push "dog" onto item 5 of animals
put animals
-- Displays "cat, , , , dog" if no other items have been
-- pushed onto this stack
```

put Command

Syntax

```
put <expression>
put <expression> into <container>
put <expression> before <container>
put <expression> after <container>
```

Description

Copies the value of an expression into, before, or after the contents of a container. If a container is not specified, the value of the expression is copied into the Command window and ToolBook shows the Command window. If ToolBook doesn't recognize the specified container, ToolBook creates a new local variable with that name and puts the value into that variable.

The `put...into` form of the command replaces the contents of `<container>` with the value o`f <expression>`. The `put...before` form of the command includes the value of `<expression>` at the beginning of the previous contents of `<container>`. The `put...after` form of the command appends the value of `<expression>` at the end of the contents of `<container>`.

If the expression evaluates to a container holding an expression, the expression in the container is not evaluated but copied literally into the container.

The contents of a container can be deleted by putting the constant `null` into the container.

Parameters

The `<expression>` parameter is any expression that yields a value. The `<container>` parameter is any expression that evaluates to a container.

Examples

In the following example, if field "Data" contains the expression `average(20,30,40)`, then:

```
put the text of field "Data" into it
   -- Puts average(20,30,40) into It (not 30)

put " not" after word 5 of text of field "Description" of\
   page 7
   -- Puts the word " not" into the text of field
   -- Description following the fifth word
```

readFile Command

Syntax

```
readFile <file name> to <character>
readFile <file name> for <number of characters>
```

Description

Reads from a specified ASCII file, starting at the beginning and continuing until ToolBook encounters the specified character or until ToolBook reads a specified number of characters. The file must have been opened with the `openFile` command. The characters that are read from the file are put into `It`. End-of-line characters, spaces, and tabs are included as characters.

When a file is first opened, its 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 just after the last character read. When ToolBook reads to a specified character, that character is discarded and not included in the text that ToolBook puts into `It`.

When ToolBook reaches the end of a file, ToolBook sets `It` to `null`, sets the `sysError` property to `end of file` and sets the `sysErrorNumber` property to `565`. To avoid an infinite loop, you must check `It`, `sysError` or `sysErrorNumber` for these values after every execution of the `readFile` command.

After ToolBook reads from a file, the file must be explicitly closed with the `closeFile` command if it is to be read again from the beginning of the file or written to in the same session.

Parameters

The `<file name>` parameter is an expression that evaluates to the name of any text-only ASCII file. The `<character>` parameter is a character specified literally or as a constant such as `quote` or `cr`. The `<number of characters>` parameter is an expression that yields a non-negative integer.

Example

```
to handle buttonUp
   openFile "donors.txt"
   while true
      readFile "donors.txt" to lf
         -- Read to first LF or CRLF
      if it is null
         -- Null value indicates end of file
         break while
      end
      put it after text of field "Name"
   end
   closeFile "donors.txt"
end buttonUp
```

remove (image) Command

Syntax

```
remove backgroundImage
remove pageImage
```

Description

Removes stored images for the current background or current page. If images have been stored for more than one display device, all stored images are removed. After issuing this command, when a copy of the book is saved using saveAs, the book's file size will decrease.

Storing a background or page image in the file can cause complex pages to display more quickly. For details about other techniques to optimize page display speed, see Chapter 3, "Tips for ToolBook Authors," in the ToolBook Ideas booklet.

See also the `store` command.

Example

```
remove backgroundImage
   -- removes stored image for the current background
```

remove menu Command

Syntax

```
remove menu <menu name> [at <level>]
```

Description

Removes a specified menu name from the menu bar and shifts other menus to the left. The shortcut keys for the menu and the menu commands on the menu are not active. If a level is specified, the menu is removed at that working level (`Reader`, `Author`, or `both`). If a level is not specified, the menu is removed at the working level in effect when the command is executed.

The `remove menu` command is useful in applications where you need to replace the standard ToolBook menus with custom menus.

For details about removing menus and menu items, see "Customizing the Menu Bar," in Chapter 6, "Beyond the Basics," of Using OpenScript.

Parameters

The `<menu name>` parameter is the name of a menu as it appears on the menu bar. If you want, you can omit leading numerals and any non-alphanumeric characters (such as spaces and punctuation, but not underscores).

The `<level>` parameter can be `author`, `reader`, or `both`; the default is the current working level.

Example

```
remove menu "Text"
   -- Removes the Text menu from the menu bar
```

remove menuItem

Command

Syntax

```
remove menuItem <menu item> [at <level>]
```

Description

Removes a menu command from a menu. If a working level is specified, the command is removed only at that level (`Reader`, `Author`, or `both`). If a level is not specified, the command is removed only at the current working level. Removing a command from a menu does not affect the default handler for that menu command.

After you remove a menu item that had accelerator keys, pressing those keys once again sends corresponding event messages, such as `keyDown`.

To put removed commands back in a menu, use the `restore menuBar` command.

For details about removing menus and menu items, see "Customizing the Menu Bar," in Chapter 6, "Beyond the Basics," of Using OpenScript.

Parameters

The `<menu item>` parameter is the name of a command as it appears in a menu. If you want, you can omit leading numerals and any non-alphanumeric characters (such as spaces and punctuation, but not underscores) so that the name matches the message that ToolBook sends when a user chooses the menu command. If the menu command to be removed is one whose name changes depending on the context (for example, the Properties and Reshape commands), `<menu item>` must be the name of the menu as it appears in the current context.

The `<level>` parameter must be `author`, `reader`, or `both`.

Examples

```
remove menuItem "Author"
   -- Removes the Author command at the current level

remove menuItem "clearall" at reader
   -- Removes the Clear All command at Reader level
```

request Command

Syntax

```
request <question> [with <reply> [or <reply2> [or <reply3>]]]
```

Description

Displays a dialog box with a question and up to three buttons, each representing a different reply. When one of the buttons is clicked, the text of that button's label goes into `It`. If the reader pressesAlt+F4 or Esc, ToolBook puts `cancel` into `It` and `sysError` and puts `566` into `sysErrorNumber`. The button labels are specified by the reply parameters (`<reply>`, `<reply2>`, and `<reply3>`). The replies can be up to approximately 8 characters long, depending on the width of the characters you specify. If a reply is not specified, ToolBook puts a button labeled OK into the dialog box. The first reply parameter appears on the left as the default button. The reader cannot continue to work in the current instance of ToolBook until one of the buttons is chosen.

Also see the `ask` command.

Parameters

The `<question>` and `<reply>` parameters are any expressions that yield strings. The expressions must be enclosed in quotation marks if they contain spaces or special characters. Each `<reply>` string can be up to 8 characters long.

Example

```
-- This handler is in the script for a Continue button.
-- Choosing More adds a page to the book; choosing
-- Print prints a page; and choosing Quit exits ToolBook
to handle buttonUp
   request "What do you want to do next?" with \
      "More" or "Print" or "Quit"
   conditions
   when it is "more"
      send newPage
   when it is "print"
      send printReport
   when it is "quit"
      send exit
   end
end
```

respondRemote Command

Syntax

```
respondRemote <response>
```

Description

This command is used by ToolBook or an object to respond to a `remoteGet` or `remoteCommand` message. This command is used for transfer of response information in Windows Dynamic Data Exchange (DDE). For details about DDE, see "Using Windows Dynamic Data Exchange" in Chapter 6, "Beyond the Basics," of Using OpenScript.

You do not need to use the `respondRemote` command unless you are explicitly handling a `remoteGet` or `remoteExecute` message. ToolBook automatically issues the `respondRemote` command when ToolBook handles these messages at the system level. Once a `respondRemote` command has been executed in a handler, ToolBook ignores any other `respondRemote` commands in the handler.

Parameter

The `<response>` parameter consists of up to nine items. The first item of the `<response>` parameter is set to one of the values described in the following table.

Parameter values passed with the respondRemote command

This value...	Means...
`OK`	ToolBook accepted the request
`Failed: Busy`	ToolBook was too busy to satisfy the request
`Failed: Denied`	ToolBook refused the request

The remaining eight items in the response comprise the `LOBYTE` of the acknowledgement status of the WM_DDE_ACK DDE message and can only have a value of `0` or `1`. The meaning of the items is described in the following table. See the DDE documentation for the appropriate application for the meaning of these items.

Meaning of respondRemote parameter items

This value...	Means...
`Item 2`	Bit 7 of the LOBYTE of WM_DDE_ACK
`Item 3`	Bit 6 of the LOBYTE of WM_DDE_ACK
`Item 4`	Bit 5 of the LOBYTE of WM_DDE_ACK
`Item 5`	Bit 4 of the LOBYTE of WM_DDE_ACK
`Item 6`	Bit 3 of the LOBYTE of WM_DDE_ACK
`Item 7`	Bit 2 of the LOBYTE of WM_DDE_ACK
`Item 8`	Bit 1 of the LOBYTE of WM_DDE_ACK
`Item 9`	Bit 0 of the LOBYTE of WM_DDE_ACK

restore menuBar

Command

Syntax

```
restore menuBar [at <level>]
```

Description

Restores the menu bar and menus to their ToolBook defaults. All user-defined menus and menu commands are removed from the menu bar. If a level is specified, the menu bar is restored at that working level (`Reader`, `Author`, or `both`). If a level is not specified, the menu bar is restored at the working level in effect when the command is executed.

Parameter

The `<level>` parameter can be `author`, `reader`, or `both`. The default is the current working level.

Example

```
to handle buttonUp
   restore menuBar
end buttonUp
```

restore system

Command

Syntax

```
restore system
```

Description

Destroys all system variables, unlinks all DLLs, and restores most system properties to their default values. The `restore system` command is useful for restoring ToolBook to a known configuration and for reclaiming memory from system variables you no longer need.

The system properties not affected by this command are:

selectedText	selection	sysLevel
selectedTextLines	sysHistory	sysMousePosition

All printer properties

The `restore system` command restores all startup properties to the startup values specified in the WIN.INI file. For example, `restore system` sets `sysBooks` to the value of `startupSysBooks`. If a script sets `startupSysBooks` to a new value, that value is remains if the system is restored.

You can protect the system variables in a book from the `restore system` command by writing `enterBook` and `leaveBook` handlers that store system variables in user-defined properties. This technique is illustrated in the second example.

Examples

This first example is a handler you might put in a book script to restore ToolBook to a known configuration:

```
to handle leaveBook     -- When a reader closes this book
   restore system       -- Restore ToolBook to default
end leaveBook           -- configuration
```

This second example shows how to save the values of system variables as book properties when leaving a book and how to restore them when entering a book.

```
-- This handler restores three system variables when
-- the book is opened
to handle enterBook
   system important1, important2, important3
   set important1 to important1 of this book
   set important2 to important2 of this book
   set important3 to important3 of this book
end enterBook

-- This handler saves the values of three system variables
-- then restores the system for any new book that is opened
to handle leaveBook
   system important1, important2, important3
   set important1 of this book to important1
   set important2 of this book to important2
   set important3 of this book to important3
```

```
   restore system
end leaveBook
```

return Command

Syntax

```
return <expression>
```

Description

Returns the result of the evaluation of a user-defined function. The `return` command can only be used within a `to get` handler that defines a function. You can also use this command to exit a `to get` handler because ToolBook does not execute any statements following the `return` command.

For details about defining your own functions, see the `to get` handler structure, or "Defining Your Own Functions" in Chapter 6, "Beyond the Basics," of Using OpenScript.

Parameter

The `<expression>` parameter is any expression.

Examples

```
-- This handler defines a function called probability
-- that calculates and returns a result based on the
-- two input parameters.
to get probability occurred,observations
   return occurred/observations
end probability

-- This handler defines a procedure for calculating the
-- value of the user-defined property, myPages, and returns
-- the value as the result
to get mypages
   local pageList
   set pageList to text of field "Page Name" of page 1 \
      of book "c:\ToolBook\mybook.tbk"
   return mypages
end mypages
```

run Command

Syntax

```
run <application> [minimized]
run <file name> [minimized]
```

Description

Launches a specified application or opens a specified file with the appropriate application. If the application cannot be found or is not a Windows-executable file, the `sysError` property is set to `no such file`, `sysErrorNumber` is set to 559 and no application is launched. If you include `minimized`, the application or file is minimized.

Using this command is similar to using the Run command from the File menu of the MS-DOS Executive, or choosing the Run command from the ToolBook File menu. Note that ToolBook launches the specified application asynchronously. That is, the application might not be completely launched before ToolBook finishes executing the current handler.

Parameters

The `<application>` parameter is any expression that evaluates to the name of any Windows application, optionally combined with the name of any file and any other parameters to be passed to the application.

The `<file name>` parameter is the name of any file that runs with a Windows application. If you don't use an extension with the file name, ToolBook tries to run the file as a ToolBook book. If you have defined file extensions in your WIN.INI file, you can just specify the name and extension of the file you want to run to launch the application. For details about the WIN.INI file, see Appendix A, "Installation Reference," in Getting Started. The path name 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.

Examples

```
run "c:\windows\clock.exe" minimized
   -- Runs and minimizes the Windows clock

run "mynotes.wri"
   -- Runs Windows Write (if installed) with the file mynotes.wri

run "notepad.exe win.ini"
   -- Runs Windows Notepad with the file WIN.INI
```

save as

Command

Syntax

```
save as <file name>,<overwrite>
```

Description

Saves the current book as a new file with a specified file name. The current book is closed without saving any changes, and the new book is opened at the same page the current book had open.

Normally, the contents of the current book replace the contents of any existing file with the specified name. If an error occurs while saving the file, the `sysError` and `sysErrorNumber` properties are set and the file is not saved.

If the specified file is currently open as a book, the `sysError` property is set to `file open as a book`, and the `sysErrorNumber` property is set to 558. If the specified file is a read-only file, the `sysError` property is set to `Read only`, and the `sysErrorNumber` property is set to 560. If the `<overwrite>` parameter is `false` and a file with the specified name already exists, then the `sysError` property is set to `File exists`, and the sysErrorNumber property is set to 627.

`Save As` compacts a book as it saves it. If you are making many changes or cutting and pasting large objects, using the `save as` command may significantly reduce your file size.

Parameters

The `<file name>` parameter is any expression that evaluates to a valid file name or a container that contains a valid file name, including the path name if necessary. The `<overwrite>` parameter must be an expression that evaluates to `true` or `false`, indicating whether an existing file should be overwritten.

Example

```
save as "c:\mybooks\oldbook.tbk",true
   -- Saves the current file using the name specified
   -- between quotes and overwrites any existing file with
   -- that name
```

save changes

Command

Syntax

```
save changes to book <book name>
```

Description

Saves any changes made to a book. The book must be open and must be named; `save changes` does not work on an untitled book.

Parameter

The `<book name>` parameter is any expression that yields the name of a currently open book.

Examples

```
save changes to this book
-- Saves changes to the current book

set text of field "OutOfDate" of page 1 of book \
   "mailabel.tbk" to true
save changes to book "mailabel.tbk"
   -- Modifies a field in a book open in another instance
   -- of ToolBook then saves the change
```

search Command

Syntax

```
search [page] for <string> [by case] [as word] [locateOnly]

search [page] excluding background for <string> [by case] [as
   word] [locateOnly]

search [page] records for <string> [by case] [as word]
   [locateOnly]

search [page] in <rfield> [ , <rfield> ] ... for <string> [by
   case] [as word] [locateOnly]

search again
```

Description

Searches for a string in the text of fields. ToolBook looks for any occurrence of the characters in the search string. The different forms of the command effect which pages and fields ToolBook will search.

The `search for ...` form of the command will search all pages and fields. The `search excluding background for ...` form of the command will not search background fields (but page and record fields will be searched). The `search records for ...` form of the command will search record fields only. Finally, the `search in <rfield>, <rfield>, ...` form of the command will search only the specified recordFields. Hidden fields are never included in a search.

Normally, when a string is found ToolBook selects the string and navigates to the page containing it. You can use the `locateOnly` keyword to search for the string without selecting it or navigating to the page that contains the string. Instead, the contents of `It` is set to a comma separated list of three items: the name of the field or record field where the string was found, followed by two integers that indicate the starting and ending characters of the matching text in that field.

If the `page` option is included, the search is restricted to the current page. If the `by case` option is included, ToolBook only finds occurrences that match the case of the search string. If the `as word` option is included, ToolBook looks only for the characters in the search string occurring as a whole word.

The `search again` command repeats the last search. If the `search again` command is executed before a search has been made, ToolBook sets the `sysError` property to `no search to repeat` and the `sysErrorNumber` property to 8112.

If ToolBook cannot find the search string, ToolBook sets the `sysError` property to `not found` and the `sysErrorNumber` property to 641.

The search string can contain spaces, so you can search for more than one word. The search string can also include quotation marks, and you can search for special characters such as tab spaces and paragraph endings, as shown in the following table.

Searching for special characters

To find...	Use...
A tab space	`^t`
The end of a paragraph	`^p`
A new-line ending created with Shift+Enter	`^n`
A caret, or circumflex symbol (^)	`^^`
A question mark	`^?`
An asterisk	`^*`
A non-breaking space	`Alt+0160`
Extended characters from Appendix A in "Using ToolBook"	`Alt+0+code`

The search string must include a caret (`^`) in front of a question mark or an asterisk if you want to search for either of these characters. This is because the question mark and asterisk can be used as wildcards in searches. Wildcards are used to represent other characters in a text string.

You can use the wildcards in the search string to help you find words spelled two different ways or to find various words with the same root. For example:

Use a question mark (`?`) to represent any single unknown or variable character in the search string. For example, the search string `?ffect` finds both `"affect"` and `"effect,"` and the search string `compl?ment` finds both `"compliment"` and `"complement."`

Use an asterisk (`*`) to represent zero or more characters in a single word in the search text. For example, the string `a*ct` finds words such as `"act,"` `"affect,"` `"abject,"` and `"character"` as well as any misspelled words that include the specified characters.

Parameter

The `<string>` parameter is a string or any expression that yields a string.

Examples

```
search for "the"
   -- Finds and selects the first consecutive
   -- occurrence of the letters the, as in
   -- Theater or together

search page records for "the" by case as word locateOnly
   -- Finds the first occurrence of the word 'the' in
   -- lowercase in a record field on the current page.
   -- If the search is successful the location of the
   -- string found is returned in the variable, It.

-- The following finds all occurrences of the word "None"
-- and change their fontStyle to bold
set sysError to null
go to the first page
search for "None" by case as word
while sysErrorNumber is not 641 -- 641 is "not found"
   send bold
   search again
end while
```

seed Command

Syntax

```
seed <number>
```

Description

Sets the starting point for the random-number generator. Seeding to the same number will always generate the same random sequence.

For details about random numbers, see the `random` function.

A good source of a number to seed the random number generator is the `sysTime` property.

Parameter

The `<number>` parameter is any expression that yields a positive integer in the range 0 to 32767.

Example

```
set sysTimeFormat to "seconds"
   -- Change format so sysTime yields a number
seed sysTime mod 32767
   -- Re-initialize the random number generator
```

select Command

Syntax

```
select <objects>
select <string specifier>
```

Description

Changes the current selection to include specified objects. The `select <objects>` form of the command is used to select a single object. You can select only the current page or objects on the current page or background.

The `select <string specifier>` form of the command is used to select text in a field or record field. The text that is selected becomes the value of the selectedText property.

Parameter

The `<objects>` parameter is an expression that evaluates to a comma-separated list of one or more object identifiers. The following types of objects can be selected:

angledLine	group	pie
arc	irregularPolygon	polygon
button	line	recordField
curve	page	rectangle
ellipse	paintObject	roundedRectangle
field	picture	

The `<string specifier>` parameter is an expression that evaluates to a chunk of text in a field or record field. For details about specifying chunks of text, see "Expressions with String Specifiers" in Chapter 2, "Script Basics," of Using OpenScript.

Examples

```
select button id 3

select text of field "Names"
   -- Selects all text in the field Names

select word 1 of textline 3 of text of recordField "Address"
   -- Selects the first word of the third line in the
   -- record field Address

select rectangle "box", ellipse "circle"
send cut
   -- Cuts a rectangle and an ellipse to the Clipboard
```

select all

Command

Syntax

```
select all [<object type>]
select all from <location> to <location>
```

Description

Selects all objects or all objects of a particular type on either a page or background. The result becomes the value of the `selection` property.

Parameters

The `<object type>` parameter is an expression that evaluates to one of the following:

angledLine	group	pie
arc	irregularPolygon	polygon
button	line	recordField
curve	paintObject	rectangle
ellipse	picture	roundedRectangle
field		

The `<location>` parameters each specify a point on the page, represented by two numbers separated by a comma. The numbers are interpreted in page units. The first number is the distance from the left edge of the screen; the second number is the distance from the top of the screen. Each `<location>` parameter can be any two expressions, separated by a comma, that each yield a number, or any single expression that yields two numbers separated by a comma.

The two points defined by the `<location>` parameters specify the top left and bottom right corners of an imaginary bounding rectangle. Any object completely inside this rectangle is included in the selection.

Examples

```
select all polygon
move selection by 1440,1440

--The following two handlers allow you to do a net selection
to handle buttonDown location
   system startLoc
   set startLoc to location
end buttonDown

to handle buttonUp location
   system startLoc
   select all from startLoc to location
end buttonUp
```

send

Command

Syntax

```
send <message> [<parameters>] [to <object>]
```

Description

Sends a message to an object. If an object is not specified, the message goes to the object sending the message and then up the object hierarchy until ToolBook finds a handler for the message. The `send` command is usually used to call another handler to perform a task for the handler sending the message, similar to calling a subroutine.

The `send` command is different from the `forward` command in that the `forward` command is usually used to pass a message that has been intercepted by a handler up the object hierarchy.

Parameters

The `<message>` parameter is any valid built-in or user-defined message. Valid messages are strings that begin with a letter and contain no spaces or punctuation marks except the underscore character. The optional `<parameters>` parameter is one or more expressions, separated by commas if there is more than one, which are sent along with the message. The `<object>` parameter is an expression that evaluates to a valid reference to an object or to `system`.

Example

```
send buttonUp to button "Next Page"
   -- Sends a buttonUp message to the button, "Next Page"
```

set Command

Syntax

```
set [the] <container> [of <object>] to <value>
```

Description

Changes the value of a specified property or other container.

Parameters

The `<container>` parameter is any expression that evaluates to a reference to a valid container. Examples of containers include variables, settable properties, and the Command window. For details about containers, see "Values and Expressions of Values" in Chapter 2, "Script Basics," of Using OpenScript.

The `<object>` parameter is any expression that evaluates to an object or window name. The `<value>` parameter is any expression or comma-separated list of expressions that evaluates to valid contents for a particular container or to a valid setting for a particular property.

Examples

```
set BowlingPins to 10
   -- Puts 10 into the variable BowlingPins

set the fontFace of field id 1 to "Tms Rmn"
   -- Sets the object's fontFace property

set the caption of the target to item 1 of argList
   -- Sets the caption property of the target to the first
   -- argument to the handler

set the script of this page to text of field \
   "Script Sample" of page "Try it yourself"
   -- Sets the script property to whatever is in the field,
   -- "Script Sample"
```

setRemote Command

Syntax

```
setRemote <item> to <value> [application <server name>] [topic
   <topic>]
```

Description

Sets the value of an item in another instance of ToolBook or in another Windows application, using Windows Dynamic Data Exchange (DDE) protocol. When you use this command to set the value of an item in another instance of ToolBook, that instance receives the `remoteSet` message. For details about DDE, see "Using Windows Dynamic Data Exchange" in Chapter 6, "Beyond the Basics," of Using OpenScript.

Immediately after ToolBook executes the `setRemote` command, the values of the `sysError` and `sysErrorNumber` properties are set to indicate the status of the command. The `sysErrorNumber` property is set to a number indicating the status. The `sysError` property is set to a list of nine items. The first item indicates the status of the remote request, as described in the following table. The last eight items describe the `LOBYTE` of the `WM_DDE_ACK DDE` message, which does not have any meaning to ToolBook but may have meaning to the application you are exchanging data with. For details about the meaning of these items, see the DDE documentation for the appropriate application.

SysErrorNumber and sysError values for the setRemote command

sysErrorNumber/ sysError	Means
8123 `OK`	The application responded and the requested data was put into `It`
8125 `Failed: Busy`	The application responded but is doing something else and failed to perform the requested action
8124 `Failed: Denied`	The application responded but could not or would not satisfy the request
8126 `Failed: Memory Error`	ToolBook did not have enough global or local memory to generate the request or accept a response
8128 `Failed: Interrupted`	The application responded, but the connection was broken before the application acknowledged the command
8127 `Failed: No Server`	No applications responded to the request

The `setRemote` command does not bypass normal password protections.

Parameters

The `<item>` parameter is any expression that evaluates to a settable item recognized by the responding application. If the responding application is ToolBook, the item is typically a settable property. The `<value>` parameter is any expression.

The `<application>` parameter is any expression that evaluates to the DDE name of a Windows application. For instance, the DDE name of Microsoft Excel is `Excel`. If there is more than one instance of the application, only the first response is processed. If the `<application>` parameter is omitted, any application that understands DDE protocol can respond.

The `<topic>` parameter is any expression that evaluates to a topic recognized by the remote instance. The topic is usually a file name. See the DDE documentation for the appropriate application for the valid topics for that application. If there is more than one instance of the specified application with the specified topic, only the first response is processed. ToolBook recognizes `(Untitled)` as a valid topic if the currently displayed book is untitled. Otherwise, ToolBook recognizes the file name of the currently open book. You can also omit the topic altogether or use the generic topic `system`, in which case any instance of a specified application can respond.

Examples

The following handler updates the date in the `DateApproved` field in another book.

```
to handle buttonUp
   setRemote "text of field DateApproved" to sysDate\
      application toolbook
end buttonUp
```

The following handler updates a phone number in an Excel spreadsheet after a reader edits the phone number in a ToolBook field named Phone. The handler is in the field's script.

```
to handle leaveField
   setRemote "R10:35" to text of field "Phone" \
      application "excel" topic "personnel.xls"
end leaveField
```

show Command

Syntax

```
show [<object>] [at <location>]
```

Description

Shows the specified object if it was previously hidden. The result of this command is the same as setting the object's `visible` property to `true`. Showing an object that is already visible has no effect.

Also see the `hide` command.

Parameters

The `<object>` parameter is an expression that evaluates to a valid identifier for one of the following objects, windows, or palettes:

Showable objects are:

angledLine	irregularPolygon	polygon
arc	line	polygonPalette
button	linePalette	recordField
colorTray	mainWindow	rectangle
commandWindow	menuBar	roundedRectangle
curve	paintObject	scrollBar
ellipse	patternPalette	spectrumPalette
field	picture	statusBox
group	pie	toolPalette

The `<location>` parameter is any two expressions separated by a comma and that each yield a number, or any single expression that yields two numbers separated by a comma. For objects, the numbers are in page units and specify the horizontal and vertical coordinates of the upper left corner of the object relative to the upper left corner of the main ToolBook window. For palettes and windows, the numbers are in screen coordinates and indicate the horizontal and vertical coordinates of the upper left corner of the window or palette in relation to the upper left corner of the screen.

Scripts that show the Command window will cause an error in Runtime ToolBook.

Examples

```
-- This handler displays a different hint field depending
-- on which incorrect answer is being reviewed
to handle wrongAnswer whichPage
   conditions
   when whichPage is 2
      show field "Hint1" of page "Review"
   when whichPage is 3
      show field "Hint2" of page "Review"
   when whichPage is 5
      show field "Hint3" of page "Review"
   when whichPage is 7
      show field "Hint4" of page "Review"
   end conditions
```

```
end
```

sort Command

Syntax

```
sort [pages <number> to <number>] [by <sort key> [,<sort key>...]]
```

Description

Sorts specified pages that share the current background. The sorted pages are reordered based on the specified sort keys.

Unlike the Sort command on the Page menu, which sorts pages only by the content of their record fields, the `sort` command can sort pages by any expression that can be evaluated in the context of a page. For example, you can use the `sort` command to sort pages based on page name or the sum of several numbers on the page.

Characters are sorted according to their ANSI values. For details about the ANSI character set, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook.

Parameters

The `<number>` parameter is an expression that evaluates to a positive integer which specifies the pages to be sorted. The default is all pages that share the current background.

Each `<sort key>` parameter consists of:

```
[<order>] <type> <sort expression>
```

The `<order>` parameter can be either `ascending` or `descending`. The default value is `ascending`. The `<type>` parameter specifies the type of sort and can be `date`, `name`, `text`, or `number`. For details about name comparison, see the `name` special term.

The `<sort expression>` parameter is any expression that can be evaluated in the context of a page; for example, the page name or ID. The default is the text of the record field with the lowest layer number on the current background. To see how ToolBook evaluates a particular sort expression, go to any page that will be included in the sort, enter the sort expression in the Command window, and press Enter.

The `<sort key>` parameter can be repeated any number of times, resulting in multiple sort keys.

Examples

```
-- This example sorts pages 1 to 10 alphabetically
-- by the content of one record field:
sort pages 1 to 10 by ascending text text of recordfield "zipcode"

-- This example shows a sort based on the content of
-- two record fields:
sort pages 10 to 20 by ascending text text of\
   recordField id 1, descending number word 2 of text of\
   recordField id 2
```

store (image) Command

Syntax

```
store backgroundImage
store pageImage
```

Description

Commands used to store a compressed image of the current background or the current page. The stored image is a device-dependent bitmap that is saved with the book. Images can be stored for different display devices by running the book on different systems and issuing this command after the page is displayed. When the background or page is displayed, ToolBook will automatically display the correct stored image for the current system.

The image will not include any objects that are drawn directly to the screen, and will increase the book's file size. Issuing this command will replace any currently stored image of the background or page for the current display device.

Storing a background or page image in the file can cause complex pages to display more quickly. For details about other techniques to optimize page display speed, see Chapter 3, "Tips for ToolBook Authors," in the ToolBook Ideas booklet.

See also the `remove` command.

Example

```
store foregroundImage
   -- Stores image for the current foreground in the
   -- ToolBook file
```

system Command

Syntax

system [variable[s]] <variable list>

Description

Makes a variable name known and its contents available to any script in the current instance of ToolBook. 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 particular handler. ToolBook initializes system variables to null when they are created.

A system variable is available to any script in the ToolBook instance in which it is defined. However, a system variable defined in one instance of a book is not available to another instance of the same book. Also, a handler cannot contain both a system variable and a local variable with the same name.

For details about system variables, see "Variables" in Chapter 2, "Script Basics," of Using OpenScript. Also see the local and restore system commands.

Parameter

The <variable list> parameter contains one or more variable names, separated by commas if there is more than one.

Example

The following handlers are in the script for a book. The handlers save the text of a field in a system variable when a reader enters the field, then checks to see if the text has changed when the reader leaves the field.

```
to handle enterField
   system fieldText
   put the text of the target into fieldText
   forward
end enterField

to handle leaveField
   system fieldText
   if fieldText is not the text of the target
      -- Check to see if the text of the field
      -- has changed
      send update
         -- Send a user-defined message
   end if
   forward
end leaveField
```

uncheck menuItem

Command

Syntax

```
uncheck menuItem <menu item> [at <level>]
```

Description

Removes a check from beside a specified user-defined menu item if it was checked. If a level is specified, the check is removed from the menu item at that working level (Reader, Author, or both). If a level is not specified, the check is removed only at the working level in effect when the command is executed. You cannot check and uncheck ToolBook's default menu items. For details about checking and unchecking menu items, see "Customizing the Menu Bar" in Chapter 6, "Beyond the Basics," of Using OpenScript.

Parameters

The `<menu item>` parameter is the name of a command as it appears on a menu. If you want, you can omit leading numerals and any non-alphanumeric characters (such as spaces and punctuation, but not underscores) so that the name matches the message that ToolBook sends when a user chooses the menu item.

The `<level>` parameter must be `author`, `reader`, or `both`.

Example

```
uncheck menuItem "Start Animation"
   -- Removes the check beside a Start Animation
   -- menu item at the current working level
```

unlinkDLL Command

Syntax

```
unlinkDLL <dll name>
```

Description

Unlinks the specified DLL and frees a reference to the DLL. When all references for a DLL are freed, the DLL and the resources it uses are freed.

Parameter

The `<dll name>` parameter is the name of the DLL that you want to unlink and is any expression which evaluates to a string.

Example

```
unlinkDLL "music.dll"
   -- Unlinks the DLL music.dll
```

unselect

Command

Syntax

```
unselect <object>
```

Description

Unselects a specified object and removes the object's name from the value of the `selection` property.

Parameters

The `<object>` parameter is any expression that yields a valid reference to an object.

Example

```
unselect irregularpolygon id 6
   -- Unselects an irregular polygon whose idNumber is 6
```

untranslateAllWindowMessages Command

Syntax

```
untranslateAllWindowMessages [for <winHandle>]
```

Description

Removes Windows-to-OpenScript message translation for the specified window.

Parameters

<winHandle> Specifies the window handle of the window to have translation removed. The default value is the handle of the main ToolBook window.

Example

```
-- Disables all Windows-to-OpenScript  translation for
-- the main window
to handle leaveBackground
   untranslateAllWindowMessages
end
```

untranslateWindowMessages Command

Syntax

```
untranslateWindowMessage <winMsg> [for <winHandle>]
```

Description

Disables Windows-to-OpenScript message translation for the specified message and window.

Parameters

<winMsg> The decimal number of the Windows message whose translation is to be removed.

<winHandle> Specifies the window handle of the window to have translation removed. The default value is the handle of the main ToolBook window.

Example

```
-- Disables Windows-to-OpenScript message translation
-- of Windows message 23 for the main ToolBook window
untranslateWindowMessage 23
```

writeFile Command

Syntax

```
writeFile <text> to <file name>
```

Description

Writes text to an open file specified by the `<file name>` parameter. 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.

After ToolBook writes to a file, the file must be explicitly closed with the `closeFile` command then reopened using the `openFile` command if it is to be read from in the same session.

Parameters

The `<text>` parameter is any expression that yields a string. The `<file name>` parameter is any expression that evaluates to a valid file name, including the path name if necessary. If you don't provide a path name, ToolBook writes the file to the current directory.

Example

```
createFile "newdata.dat"
   -- Creates and opens a file
writeFile text of field "Name" of page 1 of \
   this book to "newdata.dat"
      -- Writes text to the file
closeFile "newdata.dat"
   -- Closes the file
```

Functions

Quick reference

Alphabetical List

Arithmetic functions

abs
ceiling
evaluate
floor
random
round
sqrt
truncate

Trigonometric functions

acos
asin
atan
atan2
cos
cosh
hypotenuse
sin
sinh
tan
tanh

Logarithmic functions

exp
ln
log

Statistical functions

average
max
min
sum

Financial functions

annuityFactor
compoundFactor

String functions

ansiToChar
charCount
charToAnsi
lowercase
offset
textLineCount
uppercase
wordCount

Special functions

itemCount
keyState
menuState
objectFromPoint
pointer<type>
textFromPoint

Functions: Alphabetical list

Quick reference

abs(<number>)
acos(<number>)
annuityFactor(<rate>,<periods>)
ansiToChar(<number>)
asin(<number>)
atan(<number>)
atan2(<number1>,<number2>)
average(<list>)

ceiling(<number>)
charCount(<string>)
charToAnsi(<character>)
compoundFactor(<rate>,<periods>)
cos(<angle>)
cosh(<angle>)

evaluate(<expression>)
exp(<number>)

floor(<number>)

hypotenuse(<length>,<length>)

itemCount(<expression>)

keyState(<key>)

ln(<number>)
log(<number>,<base>)
lowercase(<string>)

max(<list>)
menuState(<menu name>) [at <level>]
min(<list>)

objectFromPoint(<location>)
offset(<string1>,<string2>)

pointer<type> (<offset>, <pointer>[,<new value>])

random(<integer>)
round(<number>)

sin(<angle>)
sinh(<angle>)
sqrt(<number>)
sum(<list>)

tan(<angle>)
tanh(<angle>)
textFromPoint(<location>)
textLineCount(<string>)
truncate(<number>)

uppercase(<string>)

wordCount(<string>)

abs

Function

Syntax

`abs(<number>)`

Description

Returns the absolute value of a number. If the number is negative, the `abs` function returns a positive value.

Parameter

The `<number>` parameter is any expression that yields a number.

Example

In the following example, if field `Change` contains the value `-465`:

```
put abs(text of field "Change") into it
   -- Puts 465 into It
```

acos Function

Syntax

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

Description

Returns the arccosine of a number. The returned value is in radians.

Parameter

The `<number>` parameter is any expression that yields a number in the range `-1` to `1`.

Example

```
put acos(.34) into it
   -- Puts 1.223879429267735 into It
```

annuityFactor Function

Syntax

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

Description

Returns a factor of the present value of an ordinary annuity to the annuity payment. The `<rate>` parameter is the interest rate per period expressed as a decimal fraction and `<periods>` is the number of periods over which the value is calculated and compounded. The formula used for the calculation is:

```
(1 - (1 + <rate>) ^ - <periods>) / <rate>
```

Parameters

The `<rate>` and `<periods>` parameters are any expressions that yield numbers.

Example

```
put 2000 into carLoanAmount
put .10/12 into periodRate
   -- A 10% interest rate divided by 12 months
put 36 into periods
   -- Number of months for the loan
   -- Calculate payments on loan
put carLoanAmount/annuityFactor(periodRate,periods) into\
   Payments
   -- Puts 64.53437438767517 into Payments
```

ansiToChar

Function

Syntax

```
ansiToChar(<number>)
```

Description

Returns the character represented by the ANSI decimal numeric value of a number. For details about the ANSI character set, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. Also see the `charToAnsi` function.

Parameter

The `<number>` parameter is any expression that yields an integer from

0 to 255, inclusive.

Example

```
put ansiToChar(90) into It
   -- Puts Z into It
```

asin Function

Syntax

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

Description

Returns the arcsine of a number. The returned value is in radians.

Parameter

The `<number>` parameter is any expression that yields a number in the range `-1` to `1`.

Example

```
put asin(.34) into it      -- Puts .3469168975271617 into It
```

atan

Function

Syntax

```
atan(<number>)
```

Description

Returns the arctangent of a number. The returned value is in radians.

Parameter

The `<number>` parameter is any expression that yields a number.

Example

```
put atan(30) into it
   -- Puts 1.53745330916649 into It
```

atan2 Function

Syntax

```
atan2(<number1>,<number2>)
```

Description

Returns the arctangent of `<number1>` divided by `<number2>`. The signs of both arguments are used to determine the quadrant of the returned value. The returned value is in radians.

Parameters

The `<number1>` and `<number2>` parameters are any expressions that yield numbers.

Example

```
put atan2(45,30) into It
   -- Puts .9827937232473292 into It
```

average Function

Syntax

```
average(<list>)
```

Description

Returns the average value of a list of numbers by dividing the sum of the numbers by the item count of the list.

Parameter

The `<list>` parameter is any expression or group of expressions that yields a sequence of numbers that are separated by commas.

Examples

```
put average(10.275,9.875,10.75) into Average_Rate
   -- Puts 10.3 into a variable

put average(text of field "Input_Data") into text of field\
   "Average"
   -- Puts the average of the contents of the Input_Data
   -- field into the Average field

put average(3,4,text of field "Data",7,12) into myData
   -- Puts the average of 3,4,7,12, and the contents
   -- of field Data into the variable myData
```

ceiling

Function

Syntax

```
ceiling(<number>)
```

Description

Returns the smallest integer that is greater than or equal to `<number>`.

Also see the `floor` function.

Parameter

The `<number>` parameter is any expression that yields a number.

Examples

```
put ceiling(-35.7) into It
   -- Puts -35 into It

put ceiling(543.93) into Approximate
   -- Puts 544 into Approximate
```

charCount Function

Syntax

`charCount(<string>)`

Description

Returns the number of characters, including spaces, in a string. A `crlf` (carriage return/linefeed) is counted as two characters.

Parameter

The `<string>` parameter is any expression that yields a string.

Example

In the following example, if field `Name` contains the name `John Brown`, followed by a carriage return/linefeed:

```
put charCount(text of field "Name") into NameLength
   -- Puts 12 into NameLength
```

charToAnsi

Function

Syntax

```
charToAnsi(<character>)
```

Description

Returns an integer representing the ANSI numeric value of a character. For details about the ANSI character set, see Appendix A, "ANSI and Other Character Sets," in Using ToolBook. Also see the `ansiToChar` function.

Parameter

The `<character>` parameter is any expression that evaluates to an ANSI character.

Examples

```
put charToAnsi("Z") into the commandWindow
   -- Displays 90 in the Command window

put charToAnsi(cr) into it
   -- Puts 13 into It
```

compoundFactor Function

Syntax

```
compoundFactor(<rate>,<periods>)
```

Description

Returns the future value of a compound-interest-bearing account. The `<rate>` parameter is the rate per period expressed as a decimal fraction, and `<periods>` is the number of periods over which the value is calculated and compounded.

Parameters

The `<rate>` and `<periods>` parameters are any expressions that yield numbers.

Example

To calculate the compound factor for an account at 10.375% for 12 months:

```
put the compoundFactor(.10375/12, 12) into text of\
   field Future_value_factor
   -- Returns 1.108828514726201
```

cos Function

Syntax

```
cos(<angle>)
```

Description

Returns the cosine of an angle. The angle is measured in radians. The formula for converting degrees to radians is `radians = degrees * (PI/180)`.

Parameter

The `<angle>` parameter is any expression that yields a number.

Example

To calculate the cosine of a 30-degree angle:

```
set radians to 30 * (pi/180)
   -- Radians is a variable
put cos(radians) into it
   -- Puts 0.8660253999169214 into It
```

cosh

Function

Syntax

```
cosh(<angle>)
```

Description

Returns the hyperbolic cosine of an angle. The angle is measured in radians. The formula for converting degrees to radians is

```
radians = degrees * (PI/180)
```

Parameter

The `<angle>` parameter is any expression that yields a number.

Example

To calculate the hyperbolic cosine of a 30-degree angle:

```
set radians to 30 * (pi/180)
   -- Radians is a variable
put cosh(radians) into it
   -- Puts 1.140238321076429 into It
```

evaluate Function

Syntax

```
evaluate (<expression>)
```

Description

Evaluates an expression and returns the value of the expression.

Parameter

The `<expression>` parameter is any expression.

Examples

```
put evaluate (1 + 3) into it
   -- Adds 1 to 3 and puts 4 into It
```

For the following example, assume that:

1) Field "Expression" contains the text:

   ```
   text of field "Data_1" + text of field "Data_2"
   ```

2) Field "Data_1 contains `450`

3) Field "Data_2 contains `25`.

```
put evaluate (text of field "Data_1") into it
   -- Adds field Data_1 to field Data_2 and puts the
   -- result into It
```

exp

Function

Syntax

```
exp(<number>)
```

Description

Returns the result of the constant `e` (2.7182818) raised to the power of the specified number. In other words, this function returns `e^<number>`.

Also see the `ln` function.

Parameter

The `<number>` parameter is any expression that yields a number.

Example

```
put exp(3) into it
   -- Puts 20.08553692318767 into It
```

floor

Function

Syntax

```
floor(<number>)
```

Description

Returns the largest integer that is less than or equal to `<number>`.

Also see the `ceiling` function.

Parameter

The `<number>` parameter is any expression that yields a number.

Examples

```
put floor(-35.7) into it
   -- Puts -36 into It

put floor(543.93) into Approximate
   -- Puts 543 into Approximate
```

hypotenuse Function

Syntax

```
hypotenuse(<length>,<length>)
```

Description

Returns the length of the hypotenuse of a right triangle given the length of the other two sides.

Parameter

The `<length>` parameter is any expression that yields a number.

Example

```
put hypotenuse(3,4) into it
   -- Puts 5 into It
```

itemCount

Function

Syntax

```
itemCount(<expression>)
```

Description

Returns the number of items in a comma-separated list. Literal expressions must be enclosed in quotation marks.

Parameter

The `<expression>` parameter is any expression that yields a list of zero or more items.

Examples

```
put itemCount("19,24,and,for,3") into Count
   -- Puts 5 into Count
send selectAll
   -- Select all objects on the page
step i from 1 to itemCount(selection)
   -- Repeat the loop for each item in the selection
   move item i of selection by 1440,1440
      -- Moves each item on the page
end step
```

keyState Function

Syntax

```
keyState(<key>)
```

Description

Returns whether the specified key is pressed. The returned value is either `down`, meaning that the key is pressed, or `up`, meaning that the key is not pressed. For details about built-in messages, see Chapter 3, "Messages and Properties," in Using OpenScript.

Parameter

The `<key>` parameter is a key constant or integer value that represents a key. Several of the key constants are for use with international versions of Windows; they are `keyKana`, `keyRomanji`, `keyZenkaku`, `keyHiraGana`, `keyKanji`, `keyConvert`, `keyNonConvert`, `keyAccept`, and `keyModeChange`. Other key constants may change for some keys from country to country so care should be taken in using this function in applications that may be used in more than one country. The `Keychar` message provides another way to intercept keystrokes and refers to characters by their ANSI code which is more consistent in different countries.

See Key constants for a list of the key constants and their integer values.

Example

```
if keyState(keyPrior) is down then
   go to previous page
      -- Go to the previous page if PgUp is pressed
end if
```

The following handler tests whether the Num Lock key is on when the focus moves to a field.

```
to handle enterField
   system numLock
      -- Create a system variable called numLock
   if keyState(keyNumLock) is "down"
      set numLock to true
         -- Set numLock to indicate state of Num Lock key
   else
      set numLock to false
   end
end enterField
```

ln

Function

Syntax

```
ln(<number>)
```

Description

Returns the natural logarithm, or base `e` logarithm, of a number.

Also see the `exp` function.

Parameter

The `<number>` parameter is any expression that yields a number.

Example

```
put ln(5) into it
   -- Puts 1.6094379124341 into It
```

log

Function

Syntax

```
log(<number>,<base>)
```

Description

Returns the logarithm of a number in a specified base.

Parameters

The `<number>` and `<base>` parameters are expressions that yield numbers.

Example

```
put log(5,2) into it
   -- Puts the base 2 logarithm of 5, or 2.321928094887362,\
   -- into It
```

lowercase

Function

Syntax

```
lowercase(<string>)
```

Description

Returns a string consisting of the lowercase equivalent of the `<string>` parameter. This function only acts on characters that have a lowercase equivalent in the ANSI character set.

Parameter

The `<string>` parameter is any expression that yields a string.

Example

In the following example, if field `Text` contains the string `"Give Me a 5-letter Word:"`

```
put lowercase(text of field "Text") into text of\
   field "Text"
   -- Puts "give me a 5-letter word:" into the field
```

max

Function

Syntax

```
max(<list>)
```

Description

Returns the highest value from a list of numbers.

Parameter

The `<list>` parameter is any expression that yields a list of numbers separated by commas.

Example

```
get max(23.7,16,45,89,-32)    -- Puts 89 into It
```

menuState Function

Syntax

```
menuState(<menu item> [at <level>])
```

Description

Returns a string indicating whether a menu item is active and checked. The possible returned values are shown in the following table.

Values returned by the menuState function

This value...	Means...
`active,checked`	The menu item is active (not dimmed or removed) and is checked
`inactive,checked`	The menu item is inactive (dimmed or removed) and is checked
`active,unchecked`	The menu item is active (not dimmed or removed) and is not checked
`inactive,unchecked`	The menu item is inactive (dimmed or removed) and is not checked

If a level is specified, the `menuState` function returns the state of the menu item at that working level. If a level is not specified, the state is returned for the current working level. If the specified menu item is not on a menu, the `menuState` function returns `null`.

Parameters

The `<menu item>` parameter is the name of a command as it appears on a menu. If you want, you can omit leading numerals and any non-alphanumeric characters (such as spaces and punctuation, but not underscores) so that the name matches the message that ToolBook sends when a user chooses the menu item.

The `<level>` parameter must be `author`, `reader`, or `both`.

Example

```
to handle Introduction
   -- Handles a message for a user-defined menu command
   if menuState(Introduction) is "active,unchecked"
      -- Check the menu command and go to the appropriate
      -- lesson book
      check menuItem "Introduction" at both
      save changes to this book
      go to book "c:\lessons\intro.tbk"
   else
      uncheck menuItem "Introduction" at both
   end if
end Introduction
```

min

Function

Syntax

```
min(<list>)
```

Description

Returns the lowest value from a list of numbers.

Parameter

The `<list>` parameter is a group of zero or more numbers separated by commas.

Example

```
put min(2.5,4,12,1,-3,8) into it
   -- Puts -3 into It
```

objectFromPoint Function

Syntax

`objectFromPoint(<location>)`

Returns the unique name of the object at a specified location on the current page or background. If more than one object occupies the specified location, `objectFromPoint` returns the unique name of the object on the top-most layer. If there is no object at the specified location on the page or background, `objectFromPoint` returns null.

Parameter

The `<location>` parameter defines a location in page coordinates (1440/inch). It can be any two numbers separated by a comma, or any expression that yields two numbers separated by a comma.

Example

If you put the following handler in a button script, the speaker will sound only when the mouse button is pressed and released over that button. If the mouse button is pressed over that button, but released elsewhere, ToolBook instead displays a message.

```
to handle buttonUp location
   -- Location is a parameter for the buttonUp message
   if objectFromPoint(location) = target
      -- Target is the button if and only if the mouse
      -- button was pressed over that button

      beep 1          -- Beep once
   else
      request "You released the mouse button" &&\
         "somewhere else."
   end if
end buttonUp
```

offset

Function

Syntax

```
offset(<string1>,<string2>)
```

Description

Returns the character position of the first occurrance of `<string1>` in `<string2>`. The returned value is `0` if the `<string1>` is not found in the `<string2>`.

Parameters

The `<string1>` and `<string2>` parameters are both expressions that yield strings.

Examples

```
put offset(space,"John Brown") into FirstName
   -- Puts 5 into FirstName
put offset("*", text of field "Definitions") into it
   -- Puts the number of characters from the beginning of
   -- the text of the field through the first * into It
```

pointer<type> Function

Syntax

pointer<type> (<offset>, <pointer>[,<new value>])

Description

Puts the value specified by <new value> into the location that is offset by <offset> bytes from the beginning of the data referenced by the <pointer> parameter. This function can also be used to get the value of data offset by <offset> bytes from the location being referenced by the <pointer> parameter.

Parameters

The <type> parameter is the literal data type that is being referenced. This parameter can have the value BYTE, INT, WORD, LONG, DWORD, FLOAT, DOUBLE, STRING, or POINTER.

The <offset> parameter is an expression that evaluates to the number of bytes offset from the beginning of the data being referenced. The <pointer> parameter is an expression that evaluates to the pointer. The <new value> parameter is an expression that evaluates to the new value for the value at the location that is offset by <offset> bytes from the beginning of the data referenced by the <pointer> parameter.

Example

```
set fileName to pointerSTRING(9,fileNamePointer)
   -- Sets the container fileName to the value of the data
   -- at the location offset 9 bytes from the location being
   -- pointed to by the pointer fileNamePointer

set oldValue to pointerCHAR(3,p,charToANSI("A"))
   -- Sets oldValue to the value of the data at the location
   -- that is offset 3 bytes from the location pointed to by
   -- pointer p, then assigns the value 65 to that location
```

random Function

Syntax

```
random(<integer>)
```

Description

Returns a random integer between 1 and a specified integer, inclusive. For details about initializing the random-number generator, see the `seed` command. The `random` function is useful for applications such as games that need to behave randomly.

Parameters

The `<integer>` parameter is any positive integer from `1` to `32767`, inclusive.

Example

```
put random(22456) into it
   -- Puts a random number between 1 and 22,456 into It
```

round

Function

Syntax

```
round(<number>)
```

Description

Returns a number rounded to the nearest integer. To truncate the decimal portion of a number without rounding, see the `truncate` function.

Parameter

The `<number>` parameter is any expression that yields a number.

Examples

In the following example, if field `Data` contains the value `10998.67`:

```
put round(text of field "Data") into it
   -- Puts 10999 into It

put round(1.5) into it
   -- Puts 2 into It

put round(1.4) into it
   -- Puts 1 into It

put round(-1.4) into it
   -- Puts -1 into It
```

sin

Function

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)`.

Parameter

The `<angle>` parameter is any expression that yields a number.

Example

To calculate the sine of a 30-degree angle:

```
set radians to 30 * (pi/180)
   -- Radians is a variable
put sin(radians) into it
   -- Puts 0.4999999999999999 into It
```

sinh

Function

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)`.

Parameter

The `<angle>` parameter is any expression that yields a number.

Example

To calculate the hyperbolic sine of a 30-degree angle:

```
set radians to 30 * (pi/180)
   -- Radians is a variable
put sinh(radians) into it
   -- Puts 0.5478534738880397 into It
```

sqrt

Function

Syntax

```
sqrt(<number>)
```

Description

Returns the square root of a number.

Parameter

The `<number>` parameter is any expression that yields a non-negative number.

Example

```
put sqrt(144) into it
   -- Puts 12 into It
```

sum Function

Syntax

```
sum(<list>)
```

Description

Returns the sum of a list of numbers.

Parameter

The `<list>` parameter is any expression that yields a group of one or more numbers separated by commas or end-of-line characters.

Examples

```
put sum(1,2,3,4) into BowlingPins
   -- Puts 10 into the variable BowlingPins

get sum(1,2,3,4,5)
   -- Puts 15 into It
```

tan

Function

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)`.

Parameters

The `<angle>` parameter is any expression that yields a number.

Example

To calculate the tangent of a 30-degree angle:

```
set radians to 30 * (pi/180)
   -- Radians is a variable
put tan(radians) into it
   -- Puts 0.5773502691896256 into It
```

tanh

Function

Syntax

```
tanh(<angle>)
```

Description

Returns the hyperbolic tangent of an angle that is measured in radians. The formula for converting degrees to radians is `radians = degrees * (PI/180)`.

Parameters

The `<angle>` parameter is any expression that yields a number.

Example

To calculate the hyperbolic tangent of a 30-degree angle:

```
set radians to 30 * (pi/180)
   -- Radians is a variable
put tanh(radians) into it
   -- Puts 0.4804727781564516 into It
```

textFromPoint

Function

Syntax

`textFromPoint(<location>)`

Description

Returns two numbers separated by a comma, indicating the text line number and character number in a field or record field at a specified location. The first number is equal to the number of text lines from the beginning of a field's text to the character at the specified location. The second number is equal to the number of characters between the character at the specified location and the first character of the text line containing the character at the specified location. For example, if a reader clicks the second character in the third text line of a field, the `textFromPoint` function returns `3,2`. If `<location>` falls on a field's border `textFromPoint` will return -1,-1.

ToolBook always evaluates this function in the context of a specific field. When ToolBook evaluates this function in a field's script, ToolBook returns a character from that field. If you use this function in the script for a field or other object to return a character from a different field, you must specify which field you want the character to come from; this is illustrated in the second example.

Parameter

The `<location>` parameter is any two expressions, separated by a comma, that each yield a number, or any single expression that yields two numbers separated by a comma, that define a location in page coordinates (1440/inch).

Examples

```
-- Put this handler in a field's script to select the
-- paragraph a reader clicks on. The field must be activated
-- for this script to work.
to handle buttonUp loc
   get item 1 of textFromPoint(loc)
   if it > 0 then
      select textLine it of my text
   end
end buttonUp

-- By putting this handler in a page's script, you can cause
-- ToolBook to delete any character the reader clicks on in
-- any activated field on that page
to handle buttonUp loc

   -- Check the reader clicked on a field
   if object of target = field
      get textFromPoint(loc) of target
      if item 1 of it > 0 then
         clear character (item 2 of it) of\
         textLine (item 1 of it) of text of target
      end
   end
end buttonUp
```

textLineCount

Function

Syntax

```
textLineCount(<string>)
```

Description

Returns the number of text lines in an expression. For details about text lines, see the `textLine` special term.

Parameter

The `<expression>` parameter is any expression that yields a string.

Example

In the following example, if field `List 1` contains six text lines:

```
get textLineCount(text of field "List 1")
   -- Puts 6 into It
```

truncate

Function

Syntax

```
truncate(<number>)
```

Description

Returns a truncated number. Everything to the right of the decimal point is removed, leaving only the integer portion to the left of the decimal point. The returned value is not rounded. To round a number, see the `round` function.

Parameter

The `<number>` parameter is any expression that yields a number.

Example

```
put truncate(468.97) into commandWindow
   -- Displays 468 in the Command window
```

uppercase Function

Syntax

```
uppercase(<string>)
```

Description

Returns a string consisting of the uppercase equivalent of the `<string>` parameter. Characters that lack an uppercase equivalent in the ANSI character set are returned unchanged.

Parameters

The `<string>` parameter is any expression that yields a string.

Example

```
set text of field "Name" to uppercase(text of field "Name")
   -- Makes all of the characters in field Name uppercase
```

wordCount

Function

Syntax

```
wordCount(<string>)
```

Description

Returns the number of words in an expression.

Parameter

The `<expression>` parameter is any expression that yields a string.

Example

```
put wordCount("The Rise and Fall of the Roman Empire") into it
   -- Puts 8 into It
```

Using DLLs with ToolBook

Quick reference

Overview
Linking a DLL and declaring its functions
Unlinking a DLL
Calling a DLL's functions from a script
Working with OpenScript, DLLs, and pointers
Executing OpenScript commands from a DLL

Standard ToolBook DLLs:
- TBKDB3.DLL
- TBKFILE.DLL
- TBKWIN.DLL

Using DLLs with ToolBook

Overview

By using DLLs (Windows Dynamic Link Libraries) with ToolBook, you can extend ToolBook's capabilities so that external code and resources can be shared by several books. Once a DLL is linked to ToolBook, you can call the DLL's functions from a script.

This appendix contains an overview of how to link a DLL to ToolBook and then call its functions from a script. In addition, this appendix contains an alphabetical reference of OpenScript DLL commands and pointer functions.

Note: This discussion is intended for advanced users with experience programming for Microsoft Windows.

Dynamic Link Libraries are separate programs that Windows applications can dynamically link to and call to perform useful tasks. You can write your own DLLs, and you can call functions from the DLLs included with ToolBook and Windows. The DLLs included with ToolBook are separate files. The DLLs included with Windows are modules that are part of the Windows executable files. For details about writing your own DLLs, see the documentation for the Microsoft Windows Software Developer's Kit.

The general steps to call a DLL's functions from OpenScript are:

1. Link the DLL to ToolBook with a handler containing the `linkDLL` Definition control structure. This declares each function you want to call in the DLL. Once the DLL is linked, you can simply call the functions in the DLL from a script in the same way you call OpenScript functions.
2. Include statements in a script to call the functions in the DLL.
3. When you no longer need to call the DLL's functions, unlink the DLL to free system resources with the `unlinkDLL` command.

For details about calling specific routines from one of the DLLs included with ToolBook, see the on-line documentation for the DLL.

Linking a DLL and declaring its functions

You link a DLL and declare its functions with a `linkDLL` control structure. When you declare the functions, you also declare their return types and parameter types. Typically, you link the DLL and declare its functions in a book's `enterBook` handler so the functions are available to the scripts in the book immediately after the book is opened. For example, the following `linkDLL` control structure links a DLL named `stat.dll` with a book and declares the function `amort`. This function has a `word` return type and has one `float` and two `word` parameter types.

```
linkDLL "stat.dll"
    word amort(float, word, word)
end linkDLL
```

Unlinking a DLL

When you no longer need the routines from a DLL, it’s a good idea to unlink the DLL to free system resources. You use the `unlinkDLL` command to unlink a DLL. Placing this command in a `leaveBook` handler ensures that system resources are freed when you close the book. For example, the following `unlinkDLL` command unlinks the DLL named STAT.DLL:

```
unlinkDLL "stat.dll"
```

Calling a DLL's functions from a script

After you have included a `linkDLL` control structure in a handler, and after ToolBook executes that handler, you can call the DLL's functions from a script. To call a DLL's function from a script, you use the function in a statement exactly as you would use any OpenScript function.

For example, the following statement calls the `amort` function located in the DLL named STAT.DLL, which was declared in a previous example. This statement passes the parameters `0.12, 15000`, and `36` to the `amort` function, gets the returned value, and places the returned value into the variable `monthPmt`.

```
set monthPmt to amort(0.12, 15000, 36)
```

Some DLL functions look like commands, but you cannot use the functions as you use OpenScript commands. You must always use DLL functions as you use OpenScript functions. For example, assume you want to set the current directory with the `setCurrentDirectory` function in the DLL named TBKFILE.DLL. You cannot use this function as follows, because it is not valid syntax for an OpenScript function:

```
to handle buttonUp
    setCurrentDirectory("c:\bin")
        -- This statement is incorrect
end buttonUp
```

Instead, you must use the function like this:

```
to handle buttonUp
    get setCurrentDirectory("c:\bin")
    -- This statement is correct
end buttonUp
```

Alternatively, you could use `setCurrentDirectory` with `put` or `set` or in any other context that is valid for an OpenScript function. This example uses `get` because it conveniently places the value returned by `setCurrentDirectory` into It.

Working with OpenScript, DLLs, and pointers

OpenScript includes several functions to work with pointers for various data types. A pointer specifies the location of a structure in global memory. You can dereference a pointer to get or set a member of the structure.

For example, the following `linkDLL` structure links the DLL named KERNEL, which is available with Microsoft Windows. This `linkDLL` structure declares the functions `openFile`, `globalAlloc`, `globalFree`, `globalLock`, and `globalUnlock` as well as the return data types and parameter types for these functions.

Note that ToolBook will not substitute Windows constants. For details about these Windows functions, see the Microsoft Windows Software Developer's Kit**.**

```
linkDLL kernel
    word fileOpen=openFile(string, pointer, word)
        -- Declare the function openFile so that the alias
        -- fileOpen can be used in scripts because openFile
        -- is a keyword in OpenScript
    word globalAlloc(word, dword)
    word globalFree(word)
    pointer globalLock(word)
    word globalUnlock(word)
end linkDLL
```

The functions declared in the previous handler give you the ability to dynamically allocate global memory from ToolBook, open the specified file, and get the file's complete path name. You could include the following statements in a script to call functions from the DLL named KERNEL:

```
set gmem to globalAlloc(0,256)
    -- Sets the value of the container gmem to the file handle
    -- returned by globalAlloc when allocating global memory as
    -- 256 bytes of fixed memory (GMEM_FIXED)

set lpofstruct to globalLock(gmem)
    -- Sets the value of the container lpofstruct to the long
    -- pointer to the memory object returned by locking global
    -- memory at location gmem

set hfile to fileOpen("filename.tbk", lpofstruct, 0)
    -- Sets the container hfile to the file handle of
    -- FILENAME.TBK. The lpofstruct parameter is a long pointer
    -- to the data structure OFSTRUCT (open file structure),
    -- which represents the open file, and 0 indicates the file
    -- is being opened to be read (OF_READ).

set xopenFile to pointerString(8, lpofstruct)
    -- Sets the container xopenFile to the complete path name of
    -- the opened file which is offset 8 bytes from the beginning
    -- of OFSTRUCT. lpofstruct is a pointer to the data structure
    -- OFSTRUCT for the file FILENAME.TBK.
    -- The path name is a zero-terminated string.
    -- pointerString is a built-in OpenScript function
```

Executing OpenScript Commands from a DLL

You can write DLLs that send messages to access OpenScript commands directly from other Windows applications. When the DLL is linked, the application can send Windows messages to the ToolBook main window to execute statements with OpenScript execute and evaluate commands. With the execute form of the message, the main window interprets lParam as a long pointer to a zero-terminated string and executes it as a set of OpenScript statements. The evaluate form of the message is similar except that the string is evaluated as an expression with the result returned to the caller.

To use this feature in a DLL, you must first register two Windows messages (the following examples are in C):

```
WORD wExecute = RegisterWindowMessage ("TBM_EXECUTE")
WORD wEvaluate = RegisterWindowMessage ("TBM_EVALUATE")
```

Also, the code must have obtained the handle for the window of each ToolBook instance that is to receive these messages.

The execute message is used as follows:

```
SendMessage (winHandleMain,wExecute,bNoErrBox,(LONG)lpCommands)
```

where winHandleMain indicates the window handle, bNoErrBox is true if sysSuspend should be set to false during command execution, and lpCommands points to a zero-terminated string containing the OpenScript statements to be executed. The message result is true if the statements execute successfully, or false if there is an error.

The evaluate message is used as follows:

```
SendMessage (winHandleMain,wEvaluate,bNoErrBox,(LONG)lpEvalBuff)
```

where bNoErrBox is true if sysSuspend should be set to false during command execution. The message result is true if the statements execute successfully or false if there is an error. The lpEvalBuff pointer must point to an instance of the following structure:

```
struct {
   LPSTR lpExpression;
   WORD wRetType;
   LPSTR lpRetValue;
   int nRetValueLength;
};
```

In this structure, lpExpression must point to a zero-terminated string containing the OpenScript expression to evaluate, wRetType identifies the type of the expected return value, lpRetValue points to a memory block where the return value should be stored, and nRetValueLength contains the length of the return value memory block if the type is a STRING. Note that if the return type is STRING and nRetValueLength is zero, lpRetValue is not used.

The value types of the wRetType field are:

0	CHAR	2	INT	4	LONG	6	FLOAT	8	POINTER
1	BYTE	3	WORD	5	DWORD	7	DOUBLE	9	STRING

STRING specifies a zero-terminated string.

When the OpenScript statements are executed or evaluated, self and target are bound to the current page and the value of It is null.

The following example uses both the execute and evaluate Windows messages. This C code first determines the number of pages in the book displayed in an instance whose main window handle is winHandleMain and then constructs a command that goes to each page.

```
typedef struct {
   LPSTR lpExpression;
   WORD wRetType;
   LPSTR lpRetValue;
   int nRetValueLength;
} EGMEVAL;

#define EVALTYPE_WORD   3
#define COMMAND  "step i from 1 to %u; go to page i; end"
void foo (WINHANDLE winHandleMain)

{
   WORD wExecute = RegisterWindowMessage("TBM_EXECUTE");
   WORD wEvaluate = RegisterWindowMessage("TBM_EVALUATE");
   WORD wPages;
   EGMEVAL     Eval;
   CHAR  Command[64];

   Eval.lpExpression = "pageCount of this book";
   Eval.wRetType = EVALTYPE_WORD;
   Eval.lpRetValue = (LPSTR)&wPages;

   if (!SendMessage(winHandleMain,wEvaluate,TRUE,
                    (LONG)(LPSTR)&Eval))
   {
      return
   }

   wspwintf(Command,COMMAND,wPages);

   SendMessage(winHandleMain,wExecute,TRUE,
               (LONG)(LPSTR)Command);
}
```

Standard ToolBook DLL extensions to OpenScript

The Dynamic Link Libraries described below are provided with ToolBook.

Before you can call the functions in a DLL from a script, you must first link the DLL to ToolBook and declare the functions you want to use with the `linkDLL` control structure. For details, see also Appendix B, "Using DLLs with ToolBook," in the Using OpenScript manual.

TBKDB3.DLL

The functions in TBKDB3.DLL allow you to work with dBASE III™ and dBASEIII+™ files from a ToolBook script. You should have some familiarity with dBASE programming in order to use the functions in these DLLs. These functions cannot be used by any Windows or OS/2 application other than ToolBook.

TBKFILE.DLL

The functions in TBKFILE.DLL allow you to copy and delete files as well as perform other file actions from a script.

TBKWIN.DLL

The functions in TBKWIN.DLL allow you to determine the characteristics of a display device, determine which fonts are available for display and printing, and set as well as get variable values in the WIN.INI file from a script.

TBKDB3.DLL function index

Files

closeAllDBFiles
closeDBFile
createDBFile
deleteDBFile
getDBFileName
openDBFile
packDBFile
selectDBFile

Records

firstDBRecord
gotoDBRecord
lastDBRecord
nextDBRecord
previousDBRecord

getDBNavigateToDeleted
getDBRecordCount
getDBRecordDeleted
getDBRecordNumber
removeDBRecords
setDBNavigateToDeleted
setDBRecordDeleted
writeDBRecord

Field

getDBFieldCount
getDBFieldName
getDBFieldPrecision
getDBFieldType
getDBFieldValue
getDBFieldWidth
setDBFieldValue

Index

checkDBIndex
closeDBIndexFile
createDBIndexFile
deselectDBIndexFile
getDBIndexExpression
getDBIndexFileName
openDBIndexFile
reindexDBFile
selectDBIndexFile

Key

findDBKey
firstDBKey
getDBKeyType
getDBKeyValue
lastDBKey
nextDBKey
previousDBKey

Tag

createDBFieldTag
freeDBFieldTag
setDBFieldTag

Error

getDBErrorString

Date

getDBDateFormat
setDBDateFormat

TBKFILE.DLL function index

Directory

createDirectory
removeDirectory
setCurrentDirectory
getCurrentDirectory

File

copyFile
fileExists
getFileAttributes
getFileDate
getFileList
getFileSize
moveFile
removeFile
setFileAttributes

Drive

getCurrentDrive
getDriveList
getFreeDiskSpace
setCurrentDrive

DOS environment

getDOSEnvironmentString

TBKWIN.DLL function index

Color model conversions

RGBtoHLS
HLStoRGB

Display capabilities

displayBitsPerPixel
displayColorPlanes
horizontalDisplayRes
horizontalDisplaySize
verticalDisplayRes
verticalDisplaySize

Coordinate conversions

clientFromPage
clientFromScreen
pageFromClient
pageFromScreen
screenFromClient
screenFromPage
xPixelsFromUnits
xUnitsFromPixels
yPixelsFromUnits
yUnitsFromPixels

Other display information

displayAspectX
displayAspectXY
displayAspectY
displayLogPixelsX
displayLogPixelsY

Fonts

displayFonts
printerFonts

Popup menu

popMenu

User Profile (WIN.INI file)

getWinIniVar
setWinIniVar

Application control

yieldApp

checkDBIndex

Function in TBKDB3.DLL

Syntax

```
checkDBIndex(<file name>)
```

Description

Checks the specified index file for accuracy against the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT checkDBIndex(STRING)
```

Parameters

`<file name>`	The index file to check and is any expression that evaluates to a valid index file name.

Returns

`1`	the index file is accurate
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-1`	if there were too many records to check
`-18`	if a bad key reference was found
`-19`	if multiple keys refer to same record
`-25`	if there is a record newer than the corresponding key
`-26`	if a key was not sorted
`-22`	if there was no key for a record
`-53`	if there is no index file with the specified name

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
to handle buttonUp
   get checkDBIndex("c:\egypt\hiero.???")
   if it <= -3
      local newFileName
      ask "Index file not accurate. Try new index file?"
      if sysError <> cancel and it <> null
         put it into newFileName
         get checkDBIndex(newFileName)
      end
   else
      request "Accurate Index file"
   end if
```

```
end buttonUp
```

clientFromPage

Function in TBKWIN.DLL

Syntax

```
clientFromPage(<PageScroll>, <Magnification>, <point|rectangle>)
```

Description

Converts a set of coordinates in ToolBook page units into coordinates in pixels relative to the top left corner of the client area of the ToolBook window The client area is defined as the working area below the menu bar, but not including the scroll bars.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING clientFromPage(STRING, INT, STRING)
```

Parameters

`<PageScroll>`	always the value of `sysPageScroll`
`<Magnification>`	always the value of `sysMagnification`
`<point|rectangle>`	either a string or an expression that evaluates to a point or a rectangle. A point is a string containing two numbers (x and y coordinates) separated by a comma. A rectangle is a string containing four numbers (coordinates of top left and bottom right of the rectangle) separated by commas.

Returns

If no error occurs, this function 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 a negative value:

`-20`	if there was not enough memory to execute the function
`-99`	if the first or last parameter is invalid

Example

```
get clientFromPage(sysPageScroll, sysMagnification,\
   bounds of rectangle "frame" of this page)
-- Find the location of rectangle "frame", in pixels,
-- relative to the top left corner of the working area of
-- the ToolBook window.
show CommandWindow at clientFromPage(sysPageScroll, \
   sysMagnification, "1200,5675")
   -- Show the command window in a specific position
   -- relative to items on the page
```

clientFromScreen

Function in TBKWIN.DLL

Syntax

```
clientFromScreen(<windowHandle>, <point|rectangle>)
```

Description

Converts a point or rectangle in pixels relative to the top left corner of the screen into a point or rectangle in pixels relative to the top left corner of the client area. The client area is defined as the working area below the menu bar, but not including the scroll bars.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING clientFromScreen(WORD, STRING)
```

Parameters

`<windowHandle>`	always the value of `sysWindowHandle`.
`<point|rectangle>`	either a string or an expression that evaluates to a point or a rectangle. A point is a string containing two numbers (x and y coordinates) separated by a comma. A rectangle is a string containing four numbers (coordinates of top left and bottom right of the rectangle) separated by commas.

Returns

If no error occurs, this function returns the coordinates of either a point or a rectangle, depending on the last parameter.

If an error occurs, it returns null and `sysError` is set to a negative value:

`-20` if there was not enough memory to execute the function

`-30` if the `<windowHandle>` parameter is invalid

`-99` if the `<point|rectangle>` parameter is invalid

Example

```
get clientFromScreen(sysWindowHandle, "0,0")
put "The top left corner of the screen is at" && it \
   && "(in pixels) relative to the top left corner of" \
   &  "the client area." into the commandWindow
```

closeAllDBFiles

Function in TBKDB3.DLL

Syntax

```
closeAllDBFiles()
```

Description

Closes all open dBASE files and index files.

To declare this function, include the following statement in the linkDLL control structure:

```
INT closeAllDBFiles()
```

Parameters

None

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

`-30` if the function failed to close all files for any other reason

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

Example

```
to handle exit
   -- Advances the record pointer to the first record
   -- before closing all files
   get firstDBRecord()
   get closeAllDBFiles()
end exit
```

closeDBFile

Function in TBKDB3.DLL

Syntax

```
closeDBFile(<file name>)
```

Description

Closes the specified dBASE file and all related index files.

To declare this function, include the following statement in the linkDLL control structure:

```
INT closeDBFile(STRING)
```

Parameters

<file name>	any expression that evaluates to the name of the dBASE file to close.

Returns

1	if the function was successful
-3	if there are too many clients for this DLL, or not enough memory
-28	if there is no dBASE file with the specified name
-29	if the index file could not be closed

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

Example

```
to handle error value, string
   request "Error (" & val & ") " & string
   get closeDBFile(fileName)
end error
```

closeDBIndexFile

Function in TBKDB3.DLL

Syntax

```
closeDBIndexFile(<file name>)
```

Description

Closes the specified dBASE index file.

To declare this function, include the following statement in the linkDLL control structure:

```
INT closeDBIndexFile(STRING)
```

Parameters

<file name>	any expression that evaluates to the name of the dBASE index file to close.

Returns

1	if the function was successful
-3	if there are too many clients for this DLL, or not enough memory
-8	if the file, the index or the system got corrupted
-53	if there is no index file with the specified name

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

Example

```
to handle error value, string
   request "Error (" & val & ") " & string
   get closeDBIndexFile(fileName)
end error
```

copyFile

Function in TBKFILE.DLL

Syntax

```
copyFile(<source file>, <destination>)
```

Description

Copies the specified file and names the copied file with the specified name. You can use `copyFile` to copy files from one disk to another.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT copyFile(STRING, STRING)
```

Parameters

`<source file>`	any expression that evaluates to the path and file name of the file to copy. If there is no match for the specified source file, `copyFile` will copy nothing to the specified destination file.
`<destination>`	either 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. This parameter is any expression that evaluates to a valid path or to a valid file name.

Returns

`1`	if the function was successful
`0`	if a undetermined error occured
`-1`	if there was a file I/O error
`-8`	if the source file could not be opened
`-9`	if the destination file could not be opened

Examples

```
get copyFile("taxhlp.tbk", "c:\finance\tax\taxhlp1.tbk")
   -- copies the file taxhlp.tbk and renames it taxhlp1.tbk

get copyFile(fileName1, fileName2)
   -- copies the file specified by fileName1 to the file
   -- specified by fileName2
```

createDBFieldTag

Function in TBKDB3.DLL

Syntax

```
createDBFieldTag(<number of fields>)
```

Description

Creates a field tag that specifies the number of fields in a dBASE file to be created with the `createDBFile` function. For details about creating a dBASE file, see `createDBFile`.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT createDBFieldTag(WORD)
```

Parameters

`<number of fields>` the number of fields the tag can contain. It can be any expression that evaluates to a valid number of fields for a dBASE file.

Returns

If no error occurs, `createDBFieldTag` returns a number that is the `<field tag number>` parameter to reference the tag in the `setDBFieldTag` function, the `createDBFile` function, and the `freeDBFieldTag` function. If an error occurs, such as not enough memory, `createDBFieldTag` returns 0.

Example

```
to handle newDBFile
   local dbFileName, noFields, fieldNames, fieldType,\
      tagNumber, fieldWidth, i
   ask "What is the name of the new DB file?"
   put it into dbFileName
   ask "How many fields do you want in the new DB file?"
   put it into noFields
   ask "Type the field names you want separated by a comma"
   put it into fieldNames
      if itemCount(fieldNames) <> noFields
         do
            request "The number of field names and the" &&\
               "number of fields do not agree"
            ask "How many fields do you want for the" &&\
               "new DB file?"
            put it into noFields
            ask "Type the field names you want" &&\
               "separated by a comma"
            put it into fieldNames
         until itemCount(fieldNames) = noFields
      end if
      -- iterate through each field name and ask for field
      -- information
      step i from 1 to itemCount(fieldNames)
```

```
         put item i of fieldNames into it
         ask "What is field type for field " & it & "?"
         put it && ", " into item i of fieldType
         put item i of fieldNames into it
         ask "What is field width for field " & it & "?"
         put it && ", " into item i of fieldWidth
      end step
      set sysError to 1
      -- get return value for tag number
      put createDBFieldTag(noFields) into tagNumber
         if tagNumber = 0
            request "Error creating field tag"
            break newDBFile
         end if
      step i from 1 to itemCount(fieldNames)
         get setDBFieldTag(tagNumber,i,item i of\
            fieldNames, item i of fieldType, item i of\
            fieldWidth, 0)
         if it <= 0
            request "Error setting field tag"
            break newDBFile
         end if
      end step
      get createDBFile(dbFileName, itemCount(fieldNames),\
         tagNumber, 1)
         if it <= 0
            request "Error creating file"
            break newDBFile
         end if
      get freeDBFieldTag(tagNumber)
         if it <= 0
            request "Error freeing field tag"
            break newDBFile
         end if
end newDBFile
```

createDBFile

Function in TBKDB3.DLL

Syntax

```
createDBFile(<file name>, <field tag number>, <preserve existing>)
```

Description

Creates and opens a dBASE file with the specified name and number of fields using the contents of the field tag data structure. The number of fields and their potential values depend on parameters passed with the `createDBFieldTag` and `setDBFieldTag` functions.

The procedure for creating a dBASE file with the dBASE DLL is as follows:

1. Create a field tag of the appropriate size with the `createDBFieldTag` function. This function's parameter indicates the number of fields the tag can contain. The return value from this function is a number used to reference the field tag when calling the `createDBFile` function and the `freeDBFieldTag` function.

   A field tag is an array of data structures used to create a dBASE file.

2. Set the contents of the field tag with the `setDBFieldTag` function. This function's parameters indicate for each field the tag number, the name, the type and the decimal precision (numeric fields only).

3. Call the `createDBFile` function with the appropriate parameters to create the dBASE file based on the contents of the field tag.

4. Free the field tag with the `freeDBFieldTag` function.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT createDBFile(STRING,WORD, WORD)
```

Parameters

`<file name>`	any expression that evaluates to the name of the dBASE file to be created.
`<field tag number>`	the return value from the `createDBFieldTag` function, which must be called before calling the `createDBFile` function.
`<preserve existing>`	if the dBASE file, <file name> exists then if this parameter is 1, ToolBook will not create the new file nor delete the existing one. If this paremeter is 0, ToolBook will delete the existing file and create a new one.

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-27`	if the dBASE file could not be created

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

Example

```
to handle newDBFile
   local dbFileName, noFields, fieldNames, fieldType
   local tagNumber, fieldWidth, i
   ask "What is the name of the new DB file?"
   put it into dbFileName
   ask "How many fields do you want in the new DB file?"
   put it into noFields
   ask "Type the field names you want separated by a comma"
   put it into fieldNames
      if itemCount(fieldNames) <> noFields
         do
            request "The number of field names and the" &&\
               "number of fields do not agree"
            ask "How many fields do you want for the" &&\
               "new DB file?"
            put it into noFields
            ask "Type the field names you want" &&\
               "separated by a comma"
            put it into fieldNames
         until itemCount(fieldNames) = noFields
      end if
      --iterate through each field name and ask for field
      --information
      step i from 1 to itemCount(fieldNames)
         put item i of fieldNames into it
         request "What is field type for field " & it & "?"
         put it && ", " into item i of fieldType
         put item i of fieldNames into it
         request "What is field width for field " & it & "?"
         put it && ", " into item i of fieldWidth
      end step
      set sysError to 1
      -- get return value for tag number
      put CreateDBFieldTag(noFields) into tagNumber
         if tagNumber = 0
            request "Error creating field tag"
            break newDBFile
         end if
      step i from 1 to itemCount(fieldNames)
         get setDBFieldTag(tagNumber,i,item i of\
            fieldNames, item i of fieldType, item i of\
            fieldWidth, 0)
         if it <= 0
            request "Error setting field tag"
            break newDBFile
         end if
      end step
      get createDBFile(dbFileName, tagNumber, 1)
         if it <= 0
            request "Error creating file"
            break newDBFile
```

```
         end if
      get freeDBFieldTag(tagNumber)
         if it <= 0
            request "Error freeing field tag"
            break newDBFile
         end if
end newDBFile
```

createDBIndexFile

Function in TBKDB3.DLL

Syntax

```
createDBIndexFile(<file name>, <sort expression>, <unique key>,
   <preserve existing>)
```

Description

Creates and opens a dBASE index file with the specified name and sort expression. In addition, defines whether each key in the index file must be unique and whether to delete an existing index file with the same name.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT createDBIndexFile(STRING, STRING, WORD, WORD)
```

Parameters

`<file name>` — any expression that evaluates to the name of the index file to create.

`<sort expression>` — any expression that evaluates to the expression that defines the index file's sort criteria. The `<sort expression>` parameter can include literal field names, constants, and operators. The result of the sort expression must be of the type numeric, logical, date, or character.

Sort expression syntax elements

Type	Values
Numeric operators	+ - * / ** ^
Character operators	+
Relational operators	= <> # < > <= >= $
Logical operators	.NOT. .OR. .AND.
Functions	CTOD; DATE; DELETED; DTOC; IIF; RECCOUNT RECNO; STR; TIME; UPPER; VAL
Constants	.T. .F.

`<unique key>` — determines whether each key in the index file must be unique and is any expression that evaluates to an integer. If the `<unique key>` parameter has a value of `1`, then the uniqueness of each key in the index file will be maintained. If the `<unique key>` parameter has a value of `0`, the keys will not be checked for uniqueness.

`<preserve existing>` — if the index file, <file name> exists then if this parameter is 1, ToolBook will not create the new file nor delete the existing one. If this paremeter is 0, ToolBook will delete the existing file and create a new one.

Returns

`1` if the function was successful

-3 if there are too many clients for this DLL, or not enough memory

-8 if there is an error in the sort expression, or if the file, the index or the system got corrupted

-12 if there is no current dBASE file; you must use openDBFile first

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

Example

```
if createDBIndexFile\
   ("temp.ndx", "LASTNAME + FIRSTN", 1, 0) = 1then
   -- You can now use this index to browse through the
   -- records in the order defined by the sort expression

else
   -- Handle error
end
```

createDirectory

Function in TBKFILE.DLL

Syntax

```
createDirectory(<directory name>)
```

Description

Creates a new directory with the specified name.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT createDirectory(STRING)
```

Parameters

`<directory name>`	the name of the directory to be created.

Returns

`1`	if the function was successful
`-3`	if the specified path was invalid
`-5`	if access was denied (for instance, if the current path name was specified)

Examples

```
get createDirectory("c:\training")
-- Creates a new directory named TRAINING

get createDirectory(dirName)
-- Creates a new directory whose name is the value of
-- the dirName container
```

deleteDBFile

Function in TBKDB3.DLL

Syntax

```
deleteDBFile(<file name>)
```

Description

Deletes the specified file and any currently open index files associated with it.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT deleteDBFile(STRING)
```

Parameters

`<file name>`	any expression that evaluates to the name of the dBASE file to be deleted.

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-20`	if the function failed because of an out of memory condition
`-28`	if there is no dBASE file with the specified name
`-29`	if the index file could not be closed

deselectDBIndexFile

Function in TBKDB3.DLL

Syntax

```
selectDBIndexFile()
```

Description

If there is a currently selected index file, deselects this file so that it no longer the current index file. The current index file determines the order in which to navigate in the current dBASE file. The index file must have been opened previously with the `openDBIndexFile` function.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT deselectDBIndexFile()
```

Parameters

None

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-12` if there is no current dBASE file; you must use `openDBFile` first

`-13` if there is no current index file; you must use use `openDBIndexFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

displayAspectX

Function in TBKWIN.DLL

Syntax

```
displayAspectX()
```

Description

Returns the relative width of a display device pixel.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT displayAspectX()
```

Parameters

None

Returns

If no error occurs, the `displayAspectX` function returns the relative width of a device pixel.

displayAspectXY

Function in TBKWIN.DLL

Syntax

```
displayAspectXY()
```

Description

Returns the relative diagonal width of a display device pixel.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT displayAspectXY()
```

Parameters

None

Returns

If no error occurs, the `displayAspectXY` function returns the relative diagonal width of a device pixel.

displayAspectY

Function in TBKWIN.DLL

Syntax

```
displayAspectY()
```

Description

Returns the relative height of a display device pixel.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT displayAspectY()
```

Parameters

None

Returns

If no error occurs, the displayAspectY function returns the relative width of a device pixel.

displayBitsPerPixel

Function in TBKWIN.DLL

Syntax

```
displayBitsPerPixel()
```

Description

Returns the number of color bits for each pixel.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT displayBitsPerPixel()
```

Parameters

None

Returns

If no errors occur, the `displayBitsPerPixel` function returns the number of color bits for each pixel.

displayColorPlanes

Function in TBKWIN.DLL

Syntax

```
displayColorPlanes()
```

Description

Returns the number of color planes for the display device.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT displayColorPlanes()
```

Parameters

None

Returns

If no errors occur, the displayColorPlanes function returns the number of color planes for the display device.

Example

```
if displayColorPlanes() = 1 then
   set s_colorScheme to BlackAndWhite
end
```

displayFonts

Function in TBKWIN.DLL

Syntax

```
displayFonts(<typeface name>)
```

Description

Returns the sizes of characters available for display for the specified typeface or for all the available typefaces.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING displayFonts(STRING)
```

Parameters

<typeface name>	`null`, if you want a list for all the fonts, or the name of the typeface for which you want the available sizes and styles. This parameter can be any expression that evaluates to a null or to a valid typeface name.

Returns

If no error occurs, the returned value depends on the parameter:

- If the parameter is `null`, the `displayFonts` function returns a string composed of textlines. Each textline is a comma-separated list containing the name of a typeface and the character sizes available for that typeface.
- If the parameter is the name of a typeface, the `displayFonts` function returns a comma-separated list containing the typeface name and the character sizes available for that typeface. However, if the named typeface is not available, it returns just the name of the typeface.

If an error occurs, the returned value is `null` and `sysError` is set to a negative value.

Example

```
if displayFonts(null) contains "Terminal" then
   set fontFace of field "viewer" to Terminal
else
   request "Terminal font not found."
end if
get displayFonts("Tms Rmn")
if it contains ",18" then
   request "OK to use a larger size?" with "Yes" or "No"
end if
```

displayLogPixelsX

Function in TBKWIN.DLL

Syntax

```
displayLogPixelsX()
```

Description

Returns the number of pixels per logical inch along the width of the display.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT displayLogPixelsX()
```

Parameters

None

Returns

If no error occurs, `displayLogPixelsX` returns the number of pixels per inch along the width of the display.

Example

```
put displayLogPixelsX() into commandWindow
```

displayLogPixelsY

Function in TBKWIN.DLL

Syntax

```
displayLogPixelsY()
```

Description

Returns the number of pixels per logical inch along the height of the display.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT displayLogPixelsY()
```

Parameters

None

Returns

If no error occurs, `displayLogPixelsY` returns the number of pixels per inch along the height of the display.

Example

```
put displayLogPixelsY() into commandWindow
```

fileExists

Function in TBKFILE.DLL

Syntax

```
fileExists(<file name>)
```

Description

Determines if the specified file exists.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT fileExists(STRING)
```

Parameters

`<file name>`	the name of the file to check for existence and is any expression that evaluates to a valid path and file name. The expression cannot contain the wildcard characters `?` and `*`. If no path is specified for the file the current directory will be searched followed by directories listed in the PATH environment variable (in order).

Returns

`1`	if the file exists
`0`	if the operation failed
`-3`	if the specified path name was invalid or specified file was not found.
`-15`	if the specified drive is invalid
`-18`	if wildcard characters were used in the specification.
`-20`	if the function failed due to an out of memory condition

Examples

```
get fileExists("c:\toolbook\help\help.tbk")
-- Determine if the file HELP.TBK exists in the
-- \TOOLBOOK\HELP directory on drive C:

get fileExists(fileName)
-- Determine if the file whose name is the value of the
-- container fileName exists
```

findDBKey

Function in TBKDB3.DLL

Syntax

```
findDBKey(<search string>)
```

Description

Searches the current index file to find the first key that matches the specified string. If ToolBook finds a match, the matching key becomes the current key and the record referenced by the key becomes the current record.

When the file is already indexed, this is the fastest way to search for a record.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT findDBKey(STRING)
```

Parameters

`<search string>`	the search string that ToolBook searches for a match in the index file.

Returns

If a matching string in the current index file was found, the `findDBKey` function returns `1` (exact find). This function can also return `2` (inexact find), `3` (on next key after next largest value), or `4` (search value larger than all other values).

If an error occurs, this function returns:

`0`	if there was an internal error
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-10`	if there is no such record, or the record was marked as deleted and the navigate to deleted records switch is `off` (see function `getDBNavigateToDeleted`)
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-13`	if there is no current index file; you must use use `openDBIndexFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

firstDBKey

Function in TBKDB3.DLL

Syntax

```
firstDBKey()
```

Description

Makes the first record that uses the first key in the current index file the current record. If the navigate to deleted switch is off, this function will skip records marked as deleted.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT firstDBKey()
```

Parameters

None

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-13`	if there is no current index file; you must use use `openDBIndexFile` first
`-15`	if the database is empty

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

firstDBRecord

Function in TBKDB3.DLL

Syntax

```
firstDBRecord()
```

Description

Makes the first record in the current dBASE file the current record. If the navigate to deleted switch is off, this function skips records marked as deleted until an undeleted record is found.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT firstDBRecord()
```

Parameters

None

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-10` if there is no such record (the database is empty or all records are deleted and the navigate to deleted switch is off)

`-12` if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
-- do lead read before looping through records
set recordcount to 0
set it to null
get firstDBRecord()
   if it <= 0
      send error it, "getting first record in dBASE file"
      break buttonUp
end if
```

freeDBFieldTag

Function in TBKDB3.DLL

Syntax

```
freeDBFieldTag(<field tag number>)
```

Description

Frees the memory used by the field tag created with the `createDBFieldTag` function.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT freeDBFieldTag(WORD)
```

Parameters

`<field tag number>` the number returned when creating the tag with the `createDBFieldTag` function and is any expression that evaluates to an existing field tag number.

Returns

`1` if the function was successful

`-7` if the field tag is invalid

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
to handle newDBFile
   local dbFileName, noFields, fieldNames, fieldType
   local tagNumber, fieldWidth, i
   ask "What is the name of the new DB file?"
   put it into dbFileName
   ask "How many fields do you want in the new DB file?"
   put it into noFields
   ask "Type the field names you want separated by a comma"
   put it into fieldNames
      if itemCount(fieldNames) <> noFields
         do
            request "The number of field names and the" &&\
                     "number of fields do not agree"
            ask "How many fields do you want for the new"&&\
                  "DB file?"
            put it into noFields
            ask "Type the field names you want separated"&&\
               "by a comma"
            put it into fieldNames
         until itemCount(fieldNames) = noFields
      end if
      -- iterate through each field name and ask for field
      -- information
      step i from 1 to itemCount(fieldNames)
```

```
         put item i of fieldNames into it
         request "What is field type for field " & it & "?"
         put it && ", " into item i of fieldType
         put item i of fieldNames into it
         request "What is field width for field " & it & "?"
         put it && ", " into item i of fieldWidth
      end step
      set sysError to 1
      -- get return value for tag number
      put createDBFieldTag(noFields) into tagNumber
         if tagNumber = 0
            request "Error creating field tag"
            break newDBFile
         end if
      step i from 1 to itemCount(fieldNames)
         get setDBFieldTag(tagNumber,i,\
            item i of fieldNames, item i of fieldType,\
            item i of fieldWidth, 0)
         if it <= 0
            request "Error setting field tag"
            break newDBFile
         end if
      end step
      get createDBFile(dbFileName, itemCount(fieldNames), \
            tagNumber, 1)
         if it <= 0
            request "Error creating file"
            break newDBFile
         end if
      get freeDBFieldTag(tagNumber)
         if it <= 0
            request "Error freeing field tag"
            break newDBFile
         end if
end newDBFile
```

getCurrentDirectory

Function in TBKFILE.DLL

Syntax

```
getCurrentDirectory(<drive letter>)
```

Description

Gets the current working directory for the specified disk drive.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getCurrentDirectory(STRING)
```

Parameters

`<drive letter>`	a one-character string containing the letter that designates the drive.

Returns

If no error occurs, the `getCurrentDirectory` function returns a string that contains the path to the current working directory, otherwise `sysError` is set to:

`-1`	if an internal error occured
`-20`	if the function failed due to an out of memory condition

Examples

```
get getCurrentDrive()
request "The current path is" && it & ":" &
getCurrentDirectory(it) & "."
```

getCurrentDrive

Function in TBKFILE.DLL

Syntax

```
getCurrentDrive()
```

Description

Gets the the current disk drive letter.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getCurrentDrive()
```

Parameters

None

Returns

If no error occurs, the `getCurrentDrive` function returns a one-character string containing the current disk drive letter, otherwise it returns null and sets `sysError` to a negative value.

Example

```
request "The current drive is" && getCurrentDrive()  & ":"
-- pops up a dialog box showing the current drive.
```

getDBDateFormat

Function in TBKDB3.DLL

Syntax

```
getDBDateFormat()
```

Description

Gets the date format currently in use by this DLL.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBDateFormat()
```

Parameters

None

Returns

If this function is successful, it returns a string containing the current date format. If an error occurs,this function returns null and `sysError` is set to one of the following values:

`-3` if there are too many clients for this DLL, or not enough memory

`-20` if there was not enough memory to execute the function

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBErrorString

Function in TBKDB3.DLL

Syntax

```
getDBErrorString(<error code>)
```

Description

Returns a string describing the error corresponding to a numeric error code.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBErrorString(INT)
```

Parameters

`<error code>`	any numeric error code returned by a function in this DLL.

Returns

This function returns a string describing the error. If an error occurs in this function the function returns `null` and `SysError` is set to one of the following values:

`-8`	if the file, the index or the system got corrupted
`-20`	if the function failed due to an out of memory condition
`-72`	if there is no string corresponding to that code

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBFieldCount

Function in TBKDB3.DLL

Syntax

```
getDBFieldCount()
```

Description

Returns the number of fields in the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT getDBFieldCount()
```

Parameters

None

Returns

If no error occurs, the `getDBFieldCount` function returns the number of fields in the current dBASE file. If an error occurs, this function returns:

`-3` if there are too many clients for this DLL, or not enough memory

`-12` if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
if getDBFieldCount() <= 0
   send error it, "getting count of fields in dBASE file"
end if
```

getDBFieldName

Function in TBKDB3.DLL

Syntax

```
getDBFieldName(<field position>)
```

Description

Returns the name of the field whose column number is specified.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBFieldName(WORD)
```

Parameters

<field position>	any expression that evaluates to the column number of the field in the current dBASE file.

Returns

If no error occurs, `getDBFieldName` returns the name of the field in the current dBASE file. If an error occurs, this function returns null and sysError is set to one of the following:

`-3` if there are too many clients for this DLL, or not enough memory

`-11` if there is no such field

`-12` if there is no current dBASE file; you must use `openDBFile` first

`-20` if the function failed because of an out of memory condition

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
-- loop through the fields in the dBASE file,
-- building background of book
step i from 1 to numFields
get getDBFieldName(i)
   if sysError is not null
      send error \
         sysError, "getting field name in dBASE file"
      break buttonUp
   end if
   -- build list of field names for this file
   put it & "," after fldnames
   -- draw and insert text into label field
   if labels is true
      draw field from 0,y to 1500, y + 300
      put it into text of the selection
   end if
   -- draw and name record field
   draw recordField from x,y to 9180, y+360
   set name of selection to it
```

```
   increment y by adv
end step
```

getDBFieldPrecision

Function in TBKDB3.DLL

Syntax

```
getDBFieldPrecision(<field name>)
```

Description

Returns the number of decimal places for the specified numeric field.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT getDBFieldPrecision(STRING)
```

Parameters

`<field name>`	any expression that evaluates to the name of the numeric field.

Returns

If no error occurs, the `getDBFieldPrecision` function returns the number of decimal places for the specified numeric field. If an error occurs, this function returns:

`-3`	if there are too many clients for this DLL, or not enough memory
`-4`	if the field name is invalid
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBFieldType

Function in TBKDB3.DLL

Syntax

```
getDBFieldType(<field name>)
```

Description

Returns the field type for the specified field name in the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT getDBFieldType(STRING)
```

Parameters

`<field name>`	any expression that evaluates to the name of the field for which you want the field type.

Returns

If no error occurs, the `getDBFieldType` function returns:

`1` if the field type is `character`

`2` if the field type is `logical`

`3` if the field type is `date`

`4` if the field type is `numeric`

`5` if the field type is `memo`

If an error occurs, this function returns:

`-3` if there are too many clients for this DLL, or not enough memory

`-4` if the field name is invalid

`-6` if the field type is invalid

`-12` if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBFieldValue

Function in TBKDB3.DLL

Syntax

```
getDBFieldValue(<field name>)
```

Description

Returns the value of the specified field for the current dBASE record and current dBASE file. This is the value as it exists in the record buffer; in other words, if you made changes to the field with `setDBFieldValue`, but did not use `writeDBRecord`, this function returns the value stored in the buffer, not the value in the file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBFieldValue(STRING)
```

Parameters

`<field name>`	any expression that evaluates to the name of the field whose value you want.

Returns

If no error occurs, the `getDBFieldValue` function returns the value of the specified field. If an error occurs, this function returns null and sysError is set to one of the following:

`-3`	if there are too many clients for this DLL, or not enough memory
`-4`	if the field name is invalid
`-6`	if the field type is invalid
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-23`	is there was an error while reading a memo file
`-20`	if the function failed because of an out of memory condition

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
while true
   increment recordCount
   step i from 1 to totalField
      set sysError to null
      get getDBFieldValue(item i of fieldNames)
      if sysError is not null
         send error sysError, "getting value of field."
         break buttonup
      end if
   put it into text of recordField (item i of fieldNames)
   end step
```

```
   -- read next
   get nextDBRecord()
   if it <= 0
      break while
   end if
   send newPage
end while
```

getDBFieldWidth

Function in TBKDB3.DLL

Syntax

```
getDBFieldWidth(<field name>)
```

Description

Returns the width of the specified field in the current dBASE record and file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT getDBFieldWidth(STRING)
```

Parameters

`<field name>`	the name of the field whose width you want and is any expression that evaluates to a valid field name in the current dBASE record and file.

Returns

If no error occurs, the `getDBFieldWidth` function returns the width in character positions for the current dBASE record and file. If an error occurs, this function returns:

`-3` if there are too many clients for this DLL, or not enough memory

`-4` if the field name is invalid

`-8` if the file, the index or the system got corrupted

`-12` if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBFileName

Function in TBKDB3.DLL

Syntax

```
getDBFileName()
```

Description

Returns the name of the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBFileName()
```

Parameters

None

Returns

If no error occurs, the `getDBFileName` function returns the path and file name of the current dBASE file. If an error occurs, this function returns null and sysError is set to one of the following:

`-3`	if there are too many clients for this DLL, or not enough memory
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-20`	if the function failed because of an out of memory condition

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBIndexExpression

Function in TBKDB3.DLL

Syntax

```
getDBIndexExpression()
```

Description

Returns the expression used to form the keys of the current index file for the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBIndexExpression()
```

Parameters

None

Returns

If no error occurs, the `getDBIndexExpression` returns the expression used to form the keys of the current index file for the current dBASE file. If an error occurs, this function returns `null` and `sysError` is set to one of the following values:

`-3` if there are too many clients for this DLL, or not enough memory

`-12` if there is no current dBASE file; you must use `openDBFile` first

`-13` if there is no current index file; you must use use `openDBIndexFile` first

`-16` if there is no expression

`-20` if the function failed because of an out of memory condition

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBIndexFileName

Function in TBKDB3.DLL

Syntax

```
getDBIndexFileName()
```

Description

Returns the name of the currently selected index file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBIndexFileName()
```

Parameters

None

Returns

If no error occurs, the `getDBIndexFileName` function returns the name of the current index file. If an error occurs, the function returns `null` and `sysError` is set to one of the following values:

`-3` if there are too many clients for this DLL, or not enough memory

`-12` if there is no current dBASE file; you must use `openDBFile` first

`-13` if there is no current index file; you must use use `openDBIndexFile` first

`-20` if the function failed because of an out of memory condition

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBKeyType

Function in TBKDB3.DLL

Syntax

```
getDBKeyType()
```

Description

Returns the type of the current key in the current index file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT getDBKeyType()
```

Parameters

None

Returns

If no error occurs, the `getDBKeyType` function returns one of three possible values for the type of the current key:

`1` if the current key type is `character`

`3` if the current key type is `date`

`4` if the current key type is `numeric`

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-12` if there is no current dBASE file; you must use `openDBFile` first

`-13` if there is no current index file; you must use use `openDBIndexFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBKeyValue

Function in TBKDB3.DLL

Syntax

```
getDBKeyValue()
```

Description

Returns the value of the current key for the current index file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDBKeyValue()
```

Parameters

None

Returns

If no error occurs, the `getDBKeyValue` function returns the value of the current key for the current index file. If an error occurs, this function returns `null` and `sysError` is set to one of the following values:

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-12` if there is no current dBASE file; you must use `openDBFile` first

`-13` if there is no current index file; you must use use `openDBIndexFile` first

`-14` if there is no current key; you must use a function such as `firstDBKey` first

`-20` if the function failed because of an out of memory condition

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBNavigateToDeleted

Function in TBKDB3.DLL

Syntax

```
getDBNavigateToDeleted()
```

Description

Returns the state of the switch that controls navigation to records marked as deleted. By default, this switch is `off`, which means that normal navigation from record to record will skip over the deleted records. The records are not actually removed from the file until it is compacted with `packDBFile`. To set the switch, use `setDBNavigateToDeleted`.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT getDBNavigateToDeleted()
```

Parameters

None

Returns

`0` if the navigate to deleted records switch is `off`

`1` if the navigate to deleted records switch is `on`

`-3` if there are too many clients for this DLL, or not enough memory

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
if getDBNavigateToDeleted is 1
   set it to "allowed"
else
   set it to "not  allowed"
end
request "Allow navigation to deleted records? It is now" &&\
   it & "."  with "No" or "Yes"
if it is "Yes"
   get setDBNavigateToDeleted(1)
else
   get setDBNavigateToDeleted(0)
end
```

getDBRecordCount

Function in TBKDB3.DLL

Syntax

```
getDBRecordCount()
```

Description

Returns the number of records in the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
LONG getDBRecordCount()
```

Parameters

None

Returns

If no error occurs, the `getDBRecordCount` function returns the number of records in the current dBASE file. If an error occurs, this function returns:

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-12` if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
to handle recordCount
   get getDBRecordCount()
      if it <= 0
         send error it, "getting record count in dBASE file"
         break recordCount
      end if
   put it into totalRecords
   while true
      increment recordCount
      step i from 1 to totalFields
         set sysError to null
         get getDBFieldValue(item i of fieldNames)
         if sysError is not null
            send error sysError, "getting value of field"
            break recordCount
         end if
         put it into \
            text of recordField(item i of fieldNames)
      end step

      -- Read the next record
```

```
      set it to null
      get nextDBRecord()
      if it <= 0
         if recordCount < totalRecords
            request "Only processed " & recordCount & \
               " out of " & totalRecords
         end if
         break while
      end if
   end while
end recordCount
```

getDBRecordDeleted

Function in TBKDB3.DLL

Syntax

```
getDBRecordDeleted()
```

Description

Determines if the current record has been marked for deletion.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT getDBRecordDeleted()
```

Parameters

None

Returns

`1`	if the current record is marked for deletion
`0`	if the current record is not marked for deletion
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-32`	if there is no current record

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDBRecordNumber

Function in TBKDB3.DLL

Syntax

```
getDBRecordNumber()
```

Description

Returns the number of the current record.

To declare this function, include the following statement in the `linkDLL` control structure:

```
LONG getDBRecordNumber()
```

Parameters

None

Returns

If no error occurs, the `getDBRecordNumber` function returns the number of the current record. If an error occurs, this function returns:

`0`	if there is no current record
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

getDOSEnvironmentString

Function in TBKFILE.DLL

Syntax

```
getDOSEnvironmentString(<tag name>)
```

Description

Returns the DOS environment value for the tag name, or returns the entire DOS environment string if no tag name is specified.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDOSEnvironmentString(STRING)
```

Parameters

`<tag name>`	the name of an environment variable, or `null`. It is not case-sensitive.

Returns

If no error occurs, the getDOSEnvironmentString function returns a string, otherwise it returns `null` and `sysError` is set to one of the following values:

`null`	if the function was successful
`-1`	if a `<tag name>` was specified, but the tag does not exist
`-20`	if there was insufficient memory to execute the function

Examples

```
--get the current path
set DOSPath to getDOSEnvironmentString("path")

-- get the entire environment string
set DOSEnvironment to getDOSEnvironmentString(null)
```

getDriveList

Function in TBKFILE.DLL

Syntax

```
getDriveList()
```

Description

Gets a list of valid drives on the current machine.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getDriveList()
```

Parameters

None

Returns

If no error occurs, the `getDriveList` function returns a `CRLF` separated list of drives for the current machine. If not enough memory is available to build the list, it returns `null` and sets `sysError` to `-20`.

Example

```
get getDriveList()
   -- Sets special variable it to a list of the valid
   -- drives on the current machine.
```

getFileAttributes

Function in TBKFILE.DLL

Syntax

```
getFileAttributes(<filename>)
```

Description

Gets the file attributes of the specified file

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getFileAttributes(STRING)
```

Parameters

`<filename>`	any expression that evaluates to the name of the file whose attributes you want.

Returns

The `getFileAttributes` function returns a string containing a character for each attribute that is true. If the returned value contains:

`R` the Read only attribute is true
`H` the Hidden attribute is true
`S` the System attribute is true
`D` the Directory attribute is true
`V` the Volume label attribute is true
`A` the Archive attribute is true

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

`-2` if the specified file was not found

`-3` if the specified path name was invalid

`-5` if access to the file was denied.

Examples

```
put getFileAttributes("c:\windows\msdos.exe") into\
   the commandWindow
   -- Returns "RHS" : This file's attributes are
   -- read only, hidden, and system

to get FileIsLocked fileName
   set sysError to null
   get getFileAttributes(fileName)

   -- file not found or access denied
   if sysError <> null then
      request "Can't get attributes of file" &&\
          filename & "."
      return false
```

```
   else
      return (it contains "R")
   end
end
```

getFileDate

Function in TBKFILE.DLL

Syntax

```
getFileDate(<filename>)
```

Description

Gets the date and time for the specified file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getFileDate(STRING)
```

Parameters

`<filename>`	the name of the file for which you want the date and is any expression that evaluates to a valid path and file name.

Returns

If no error occurs, the `getFileDate` function returns a string containing the file's date and time. If there is an error, `sysError` is set to one of the following values:

`-2` if the file was not found

`-3` if the path was not found

`-20` if the function failed due to an out of memory condition

Examples

```
get getFileDate("c:\toolbook\toolbook.exe")
   -- Sets if to the date of the file TOOLBOOK.EXE

get getFileDate(fileName)
   -- Sets it to the date of the file whose name is
   -- the value of fileName
```

getFileList

Function in TBKFILE.DLL

Syntax

```
getFileList(<file name>)
```

Description

Returns a list of files matching a file path and name specification.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getFileList(STRING)
```

Parameters

`<file name>`	any expression that evaluates to the name of the file to find. The expression can contain the wildcard characters `?` and `*`.

Returns

If no error occurs and if ToolBook finds files that match the `<file name>` parameter, the `getFileList` function returns a `CRLF` separated list of matching files. If an error occurs, the function returns null and `sysError` is set to one of the following values:

`-2`	if the file name was invalid
`-3`	if the path was not found
`-18`	if there was no matching file
`-20`	the function failed due to an out of memory condition

Examples

```
get getFileList("c:\toolbook\dbms*.tbk")
-- Sets it to a list of matching files

get getFileList(fileName)
-- Sets it to a CRLF separated list of matching files
```

getFileSize

Function in TBKFILE.DLL

Syntax

```
getFileSize(<file name>)
```

Description

Gets the size of the specified file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
LONG getFileSize(STRING)
```

Parameters

`<file name>`	any expression that evaluates to the name of the file for which you want the size.

Returns

If no error occurs, the `getFileSize` function returns the file's size in bytes. Otherwise it returns:

`-2` if the specified file was not found

`-3` if the specified path was not found

`-20` if the function failed due to an out of memory condition.

Examples

```
get getFileSize("c:\maps\egypt.tbk")
-- Sets it to the file size in bytes of the file EGYPT.TBK

get getFileSize(fileName)
-- Sets it to the file size in bytes of the file whose
-- name is the value of fileName
```

getFreeDiskSpace

Function in TBKFILE.DLL

Syntax

```
getFreeDiskSpace(<disk drive>)
```

Description

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

To declare this function, include the following statement in the `linkDLL` control structure:

```
LONG getFreeDiskSpace(STRING)
```

Parameters

`<disk drive>`	any expression that evaluates to the letter designating the disk drive for which you want the number of free bytes.

Returns

If no error occurs, the `getFreeDiskSpace` function returns the number of free bytes on the specified disk. Otherwise, this function returns a negative number.

Example

```
get getFreeDiskSpace("C")
-- Sets it to the number of free bytes on disk C.
```

getWinIniVar

Function in TBKWIN.DLL

Syntax

```
getWinIniVar(<section name>, <item name>)
```

Description

Returns the value of the specified item in the specified section of the `WIN.INI` file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING getWinIniVar(STRING, STRING)
```

Parameters

`<section name>`	the name of the section in the `WIN.INI` file that contains the item whose value you want. The `<section name>` parameter is any expression that evaluates to a valid `WIN.INI` section name.
`<item name>`	the name of the item whose value you want and is any expression that evaluates to a valid item name for the specified section name.

Returns

If no error occurs, the `getWinIniVar` function returns the value of the specified item in the specified section in the `WIN.INI` file, otherwise:

`-20`	if there was an out of memory condition error while trying to execute the function.

Example

```
set s_NoColors to \
   getWinIniVar("ToolBook", "startUpSysColors") is "false"
```

gotoDBRecord

Function in TBKDB3.DLL

Syntax

```
gotoDBRecord(<record number>)
```

Description

Makes the specified record the current record. In other words, navigates to the specified record number in the data file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT gotoDBRecord(DWORD)
```

Parameters

`<record number>`	the record number of the record you want to make the current record. If the value of the `<record number>` parameter is greater than the number of records in the dBASE file, then the last record becomes the current record. If the value of the `<record number>` parameter is less than or equal to `0`, then the first record becomes the current record.

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-10` if there is no such record

`-12` if there is no current dBASE file; you must use `openDBFile` first

`-87` the record was marked as deleted and the navigate to deleted records switch is `off` (see function `getDBNavigateToDeleted`)

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

HLStoRGB

Function in TBKWIN.DLL

Syntax

```
HLStoRGB(<hue>, <lightness>, <saturation>)
```

Description

Converts the specified HLS color value into an RGB color value.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING HLStoRGB(DOUBLE,DOUBLE,DOUBLE)
```

Parameters

`<hue>`	any expression that evaluates to a hue value in the range `0` to `360`.
`<lightness>`	any expression that evaluates to a lightness value in the range `0` to `100`.
`<saturation>`	any expression that evaluates to a saturation value in the range `0` to `100`.

Note: If any parameter is out of range, the function will assume a value of `0` for it. Parameters do not have to be integer numbers.

Returns

If no error occurs, the `HLStoRGB` function returns a string of three values that represent the red, green, and blue values, otherwise:

`-20`	if there was an out of memory condition error while trying to execute the function.

Example

```
set Rgb to HSLtoRGB(275.95, 75.21, 23)
```

horizontalDisplayRes

Function in TBKWIN.DLL

Syntax

```
horizontalDisplayRes()
```

Description

Returns the physical display's horizontal size in pixels.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT horizontalDisplayRes()
```

Parameters

None

Returns

The `horizontalDisplayRes` function returns the horizontal display size in pixels.

Example

```
if horizontalDisplayRes() is 640 and \
   verticalDisplayRes() is 480 then
   set s_CRT to "Standard VGA"
end if
```

horizontalDisplaySize

Function in TBKWIN.DLL

Syntax

```
horizontalDisplaySize()
```

Description

Returns the physical display's horizontal size in millimeters.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT horizontalDisplaySize()
```

Parameters

None

Returns

The `horizontalDisplaySize` function returns the horizontal display size in millimeters.

lastDBKey

Function in TBKDB3.DLL

Syntax

```
lastDBKey()
```

Description

Makes the last key in the current index file the current key and makes the record referenced by the last key the current record. Unless the navigate to deleted switch is on, this function will skip records marked as deleted.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT lastDBKey()
```

Parameters

None

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-13`	if there is no current index file; you must use use `openDBIndexFile` first
`-15`	if the database is empty

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

lastDBRecord

Function in TBKDB3.DLL

Syntax

```
lastDBRecord()
```

Description

Makes the last record in the current dBASE file the current record. If the last record is marked as deleted and the navigate to deleted switch is off, the last undeleted record becomes the current record.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT lastDBRecord()
```

Parameters

None

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-10`	if there is no such record (the database is empty or all records are deleted and the navigate to deleted switch is off.)
`-12`	if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

moveFile

Function in TBKFILE.DLL

Syntax

```
moveFile(<file spec>, <destination>)
```

Description

Move the specified file and optionally rename the moved file. With the `moveFile` function, you cannot move a file from one disk drive to another.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT moveFile(STRING, STRING)
```

Parameters

`<file spec>`	any expression that evaluates to the name of the file to be moved.
`<destination>`	any expression that evaluates to the path and, optionally, the file name where you want to move the file. If no file name is supplied, the file is named the same as `<file spec>`.

Returns

`1`	if no error occurred
`-2`	if the specified file was not found
`-3`	if the specified path name was invalid
`-5`	if access to the file was denied
`-17`	if the specified paths for the source and destination files refer to different disk drives.

Example

```
get moveFile("c:\toolbook\train1.tbk", \
   "c:\archive\traingh.tbk")
   -- Moves the file TRAIN1.TBK to the ARCHIVE directory
   -- and renames the file to TRAINGH.TBK
```

nextDBKey

Function in TBKDB3.DLL

Syntax

```
nextDBKey()
```

Description

Makes the next key in the current index file the current key and makes the record referenced by the next key the current record. If the current key is the last key, then the current key will remain the current key. Unless the navigate to deleted switch is on, this function will skip records marked as deleted.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT nextDBKey()
```

Parameters

None

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-12` if there is no current dBASE file; use `openDBFile`

`-13` no current index file; use `openDBIndexFile`

`-70` if the key was already the last key

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

nextDBRecord

Function in TBKDB3.DLL

Syntax

```
nextDBRecord()
```

Description

Makes the next record in the current dBASE file the current record. Unless the navigate to deleted switch is on, this function skips records marked as deleted.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT nextDBRecord()
```

Parameters

None

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-10` if there is no such record (current record is last record)

`-12` if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
to handle recordCount
   get getDBRecordCount()
   if it <= 0
      send error it, "getting record count in dBASE file"
      break recordCount
   end
   put it into totalRecords
   while true
      increment recordCount
      step i from 1 to totalFields
         set sysError to null
         get getDBFieldValue(item i of fieldNames)
         if sysError is not null
            send error sysError, "getting value of field"
            break recordCount
         end
      end
      put it into text of recordField(item i of fieldNames)
      -- Read the next record
      set it to null
      get nextDBRecord()
```

```
      if it <= 0
         if recordCount < totalRecords
            request "Only processed" && recordCount &&\
               "out of" && totalRecords
         end
      end
      break while
   end while
end recordCount
```

openDBFile

Function in TBKDB3.DLL

Syntax

```
openDBFile(<file name>)
```

Description

Open and initialize the specified file and make it the current dBASE file.

To declare this function, include the following statement in the linkDLL control structure:

```
INT openDBFile(STRING)
```

Parameters

<file name>	any expression that evaluates to the name of the dBASE file to be opened.

Returns

1	if the function was successful
-3	if there are too many clients for this DLL, or not enough memory
-28	if there is no dBASE file with the specified name
-88	if there is a memo field but the memo file could not be opened.

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

Example

```
ask "Name of dBASE file to open?"
put it into text of field "dBASEFileName"
get openDBFile(text of field "dBASEFileName")
if it <> 1
   send error it, "Opening dBASE file"
   break
end if
```

openDBIndexFile

Function in TBKDB3.DLL

Syntax

```
openDBIndexFile(<file name>)
```

Description

Opens the specified index file and makes it the current index file for the current dBASE file. The first logical record in the order defined by the index becomes the current record.

To declare this function, include the following statement in the linkDLL control structure:

```
INT openDBIndexFile(STRING)
```

Parameters

<file name>	any expression that evaluates to the name of the index file to be opened for the current dBASE file.

Returns

1	if the function was successful
-3	if there are too many clients for this DLL, or not enough memory
-8	if the file, the index or the system got corrupted
-12	if there is no current dBASE file; you must use openDBFile first

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

packDBFile

Function in TBKDB3.DLL

Syntax

```
packDBFile()
```

Description

Packs the current dBASE file by reclaiming space occupied by records marked for deletion. After packing, the current record is the last record in the data file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT packDBFile()
```

Parameters

None

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-31`	if the data packing operation failed for any other reason

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

pageFromClient

Function in TBKWIN.DLL

Syntax

```
pageFromClient(<PageScroll>, <Magnification>, <point|rectangle>)
```

Description

Converts a set of coordinates in pixels relative to the top left corner of the client area of the ToolBook window into coordinates in ToolBook page units. The client area is defined as the working area below the menu bar, but not including the scroll bars.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING pageFromClient(STRING, INT, STRING)
```

Parameters

`<PageScroll>`	is always the value of `sysPageScroll`.
`<Magnification>`	is always the value of `sysMagnification`.
`<point|rectangle>`	is either a string or an expression that evaluates to a point or a rectangle. A point is a string containing two numbers (x and y coordinates) separated by a comma. A rectangle is a string containing four numbers (coordinates of top left and bottom right of the rectangle) separated by commas.

Returns

If no error occurs, this function returns the coordinates of either a point or a rectangle, depending on the last parameter.

If an error occurs, it returns `null` and `sysError` is set to a negative value:

`-20` if there was not enough memory to execute the function

`-99` if the first or last parameter is invalid

Example

```
get PageFromClient(sysPageScroll, sysMagnification, \
   bounds of  mainWindow)
   -- Find where the actual boundaries of the ToolBook
   -- windows are, expressed in page units relative to
   -- to the page currently displayed.
```

pageFromScreen

Function in TBKWIN.DLL

Syntax

```
pageFromScreen(<WindowHandle>, <PageScroll>, <Magnification>,
   <point|rectangle>)
```

Description

Converts a set of coordinates in pixels relative to the the top left corner of display screen into coordinates in ToolBook page units.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING pageFromScreen(WORD, STRING, INT, STRING)
```

Parameters

`<WindowHandle>`	is always the value of `sysWindowHandle`.
`<PageScroll>`	is always the value of `sysPageScroll`.
`<Magnification>`	is always the value of `sysMagnification`.
`<point|rectangle>`	is either a string or an expression that evaluates to a point or a rectangle. A point is a string containing two numbers (x and y coordinates) separated by a comma. A rectangle is a string containing four numbers (coordinates of top left and bottom right of the rectangle) separated by commas.

Returns

If no error occurs, this function returns the coordinates of either a point or a rectangle, depending on the last parameter.

If an error occurs, it returns `null` and `sysError` is set to a negative value:

`-20`	if there was not enough memory to execute the function
`-30`	if the `<windowHandle>` parameter is invalid
`-99`	if the second or last parameter is invalid

Example

```
get pageFromScreen(sysWindowHandle, sysPageScroll, \
   sysMagnification, bounds of  mainWindow)
   -- Find where the actual boundaries of the ToolBook
   -- windows are, expressed in page units relative to
   -- the display screen
```

popMenu

Function in TBKWIN.DLL

Syntax

```
popMenu(<WindowHandle>, <PageScroll>, <Magnification>,
   <position>,<MenuItems>,<spare>)
```

Description

Displays a popup menu and returns the number of the item chosen, or zero if the user dismissed the menu without choosing an item. This menu can be activated only with the mouse.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT popMenu(WORD, STRING, INT, STRING, STRING, STRING)
```

Parameters

<WindowHandle>	is always the value of `sysWindowHandle`.
<PageScroll>	is always the value of `sysPageScroll`
<Magnification>	is always the value of `sysMagnification`
<position>	is a string or an expression that evaluates to a point. A point is a string containing two numbers (x and y coordinates) separated by a comma. This argument defines the position of the top left corner of the menu. If position is such that part of the menu would be off the screen, Windows will adjust this position to attempt to keep the menu visible on the screen.
<MenuItems>	is a comma-separated or CRLF- separated list of the items that should appear in the menu. The following symbols and characters have special meaning:

Special symbols for menu items

Symbol	meaning
`tab`	(the single character with the openScript value `tab`) causes the rest of the item to be tabbed to the right of the menu
`&`	causes the next character to be underlined
`|`	as leading character of the menu item or immediately following `{`, causes the item to be displayed in a new column separated by a vertical line; cannot be used with `\`
`\`	as leading character of the menu item or immediately following `{`, causes the item to be displayed in a new column without a vertical line; cannot be used with |
`(`	as leading character of the menu item or immediately following `|`, `\`, or `{`, disables the item; may be followed by `*` to check the item as well

`*`	as leading character of the menu item or immediately following `{`, `|`, `\` or `(`, checks the item
{	as leading character of the menu item, makes the item the parent of a submenu; subsequent items will be displayed in a submenu; use } to indicate the end of the submenu
`}`	as the only character in an item, marks the end of a submenu
`null`	a `null` menu item shows as a separator

The order in which you can use `|`,`\`,`(`,`*` or `{` is as follows:

```
["{"]["|" | "\"]["("]["*"]
```

To include one of the special characters in the text of the item, double the character. For instance, `"*Test"` will show as `"Test"` with a checkmark, but `"**Test"` will show as `"*Test"`.

`<spare>` is always `null` in version 1.0 of this DLL.

Returns

If no error occurs, this function returns the number of the chosen item, or `0` if no item was chosen. Every item in `<MenuItems>` is assigned a number in sequence; however, the function will never return the number of items that cannot actually be chosen. For instance, a separator gets a number, but this number will never be returned by the function.

If an error occurs, `popMenu` returns a negative number. Errors can be caused by an error in the syntax of `MenuItems` parameter, by an error in any of the other parameters, or by a system condition that prevents the creation or display of the menu.

Example

```
-- Display a popup menu with 2 standard items, one
-- separator, a disabled item, a checked item, and a
-- disabled as well as checked item.  Put the text  of the
-- chosen item in the command window.
to handle buttonDown
    set menuList to \
        "Choice 1," & \
        "Choice 2," & \
        "," & \
        "(Disabled Choice," & \
        "*Checked Choice," & \
        "(*Disabled And Checked Choice"
    get popMenu(sysWindowHandle,\
      sysPageScroll,sysMagnification, \
      position of rectangle "MenuMark", menuList, null)
    if it <> 0
        put item it of menuList
    else
        put "Nothing chosen"
    end
```

```
end

-- Display a hiearchical menu
to handle buttonDown
    set menuList to \
        "{Level 1 Choice 1," & \
        "SubMenu 1 Level 2 Choice 1," & \
        "SubMenu 1 Level 2 Choice 2," & \
        "}," & \
        "{Level 1 Choice 2," & \
        "SubMenu 2 Level 2 Choice 1," & \
        "SubMenu 2 Level 2 Choice 2," & \
        "}"
    put popMenu(sysWindowHandle,\
      sysPageScroll,sysMagnification, \
        position of rectangle "MenuMark", menuList,null)
end

-- Display a menu for which the description is stored in a
--field with each item on a separate line
to handle buttonDown
    put popMenu(sysWindowHandle, \
         sysPageScroll,sysMagnification, \
            position of rectangle "MenuMark", \
            text of field "menuList", null)
end
```

previousDBKey

Function in TBKDB3.DLL

Syntax

```
previousDBKey()
```

Description

Makes the key before the current key in the current index file the current key. Unless the navigate to deleted switch is on, this function will skip records marked as deleted.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT previousDBKey()
```

Parameters

None

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDABFile` first
`-13`	if there is no current index file; you must use use `openDBIndexFile` first
`-71`	if the key was already the first key

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

previousDBRecord

Function in TBKDB3.DLL

Syntax

```
previousDBRecord()
```

Description

Makes the record before the current record in the current dBASE file the current record. If the current record is the first record, then it will remain the current record. Unless the navigate to deleted switch is on, this function skips records marked as deleted.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT previousDBRecord()
```

Parameters

None

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

`-8` if the file, the index or the system got corrupted

`-10` if there is no such record (the current record is the first record)

`-12` if there is no current dBASE file; you must use `openDBFile` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
to handle buttonUp
   local i
   set sysCursor to 4
   -- step control structure advances the record pointer
   step i from 1 to 100
      get previousDBRecord()
   end step
   send displayRecord
end buttonUp
```

printerFonts

Function in TBKWIN.DLL

Syntax

```
printerFonts(<typeface name>)
```

Description

Returns the sizes of characters available for printing for the specified typeface.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING printerFonts(STRING)
```

Parameters

`<typeface name>`	parameter is either null or the name of the typeface for which you want the available sizes and styles. This parameter is any expression that evaluates to null or to a valid typeface name.

Returns

If no error occurs, the returned value depends on the parameter:

If the parameter is `null`, `printerFonts` returns a string composed of textlines. Each textline is a comma-separated 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 comma-separated list containing the typeface name and the character sizes available for that typeface. However, if the named typeface is not available, it returns just the name of the typeface.

If an error occurs, the returned value is `null` and `sysError` is set to a negative value.

Example

```
if not (PrinterFonts(null) contains "Roman") then
   request "The Roman font is not available for" &&\
      "printing. Substituting Times Roman"
end
```

reindexDBFile

Function in TBKDB3.DLL

Syntax

```
reindexDBFile(<file name>)
```

Description

Reindexes the specified index file. The data file associated with the index file must be open before calling the function.

On completion of the reindexing, the specified index file becomes the current index.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT reindexDBFile(STRING)
```

Parameters

`<file name>`	any expression that evaluates to the name of the index file to be reindexed.

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-53`	if there is no index file with the specified name

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

removeDBRecords

Function in TBKDB3.DLL

Syntax

```
removeDBRecords(<start number>, <end number>)
```

Description

Removes the records in the specified range, inclusive, from the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT removeDBRecords(DWORD, DWORD)
```

Parameters

`<start number>`	any expression that evaluates to the beginning of the range of records you want to delete.
`<end number>`	any expression that evaluates to the end of the range of the records you want to delete. To delete only one record, specify that record number for both the `<start number>` and `<end number>` parameters.

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use openDBFile first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

removeDirectory

Function in TBKFILE.DLL

Syntax

```
removeDirectory(<path name>)
```

Description

Deletes the specified directory.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT removeDirectory(STRING)
```

Parameters

`<path name>`	any expression that evaluates to the path of the directory to be deleted.

Returns

`1`	if the directory was successfully deleted
`-3`	if the specified path name was invalid or if there are files in the directory
`-5`	if access to the directory was denied
`-16`	if the current directory was specified

Examples

```
get removeDirectory("c:\temp")
   -- Delete the directory named TEMP in the root directory
   -- of drive C:

get removeDirectory(dirName)
   -- Delete the directory whose name is the value of
   --the container dirName
```

removeFile

Function in TBKFILE.DLL

Syntax

```
removeFile(<filename>)
```

Description

Deletes the specified file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT removeFile(STRING)
```

Parameters

`<filename>`	any expression that evaluates to the name of the file to delete.

Returns

`1`	if the specified file was successfully deleted
`-2`	if the specified file was not found
`-3`	if the specified path name was invalid
`-5`	if access to the file was denied

Examples

```
get removeFile("d:\salary.tbk")
   -- Delete the file named SALARY.TBK in the root directory
   --  of drive D:

get removeFile(fileName)
   -- Delete the file whose name is the value of the
   -- container fileName
```

RGBtoHLS

Function in TBKWIN.DLL

Syntax

```
RGBtoHLS(<red>, <green>, <blue>)
```

Description

Converts the specified RGB color value into an HLS color value.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING RGBtoHLS(WORD,WORD,WORD)
```

Parameters

`<red>`	any expression that evaluates to the numeric value of the red component.
`<green>`	any expression that evaluates to the numeric value of the green component.
`<blue>`	any expression that evaluates to the numeric value of the blue component.

Returns

If no error occurs, the `RGBtoHLS` function returns a string of three values that represent the hue, lightness, and saturation values, otherwise:

`-20` if there was an out of memory condition error while trying to execute the function.

screenFromClient

Function in TBKWIN.DLL

Syntax

```
screenFromClient(<windowHandle>, <point|rectangle>
```

Description

Converts a point or rectangle in pixels relative to the top left corner of the ToolBook client area into a point or rectangle in pixels relative to the top left corner of the screen. The client area is defined as the working area below the menu bar, but not including the scroll bars.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING screenFromClient(WORD, STRING)
```

Parameters

`<windowHandle>`	is always the value of sysWindowHandle.
`<point|rectangle>`	is either a string or an expression that evaluates to a point or a rectangle. A point is a string containing two numbers (x and y coordinates) separated by a comma. A rectangle is a string containing four numbers (coordinates of top left and bottom right of the rectangle) separated by commas.

Returns

If no error occurs, this function returns the coordinates of either a point or a rectangle, depending on the last parameter.

If an error occurs, it returns `null` and `sysError` is set to a negative value:

`-20` if there was not enough memory to execute the function

`-30` if the `<windowHandle>` parameter is invalid

`-99` if the `<point|rectangle>` parameter is invalid

Example

```
get screenFromClient(sysWindowHandle, "0,0")
request  "The top left corner of the client area is at" && \
   it && "(in pixels) relative to the top left corner " && \
   " of the screen."
```

screenFromPage

Function in TBKWIN.DLL

Syntax

```
screenFromPage(<WindowHandle>, <PageScroll>, <Magnification>,
   <point|rectangle>)
```

Description

Converts a set of coordinates in ToolBook page units into coordinates in pixels relative to the top left corner of the display screen.

To declare this function, include the following statement in the `linkDLL` control structure:

```
STRING screenFromPage(WORD, STRING, INT, STRING)
```

Parameters

`<WindowHandle>`	is always the value of `sysWindowHandle`.
`<PageScroll>`	is always the value of `sysPageScroll`.
`<Magnification>`	is always the value of `sysMagnification`.
`<point|rectangle>`	is either a string or an expression that evaluates to a point or a rectangle. A point is a string containing two numbers (x and y coordinates) separated by a comma. A rectangle is a string containing four numbers (coordinates of top left and bottom right of the rectangle) separated by commas.

Returns

If no error occurs, this function returns the coordinates of either a point or a rectangle, depending on the last parameter.

If an error occurs, it returns `null` and `sysError` is set to a negative value:

`-20`	if there was not enough memory to execute the function
`-30`	if the window handle was invalid
`-99`	if the second or last parameter is invalid

Example

```
-- Sets the screen location of another ToolBook instance to
-- match a graphic in this instance.
get screenFromPage(sysWindowHandle, sysPageScroll, \
   sysMagnification, bounds of rectangle "placeHolder")
executeRemote "Set bounds of mainWindow to" && it \
   application "ToolBook" topic "Inset.tbk"
```

selectDBFile

Function in TBKDB3.DLL

Syntax

```
selectDBFile(<file name>)
```

Description

Makes the specified file the current dBASE file. This function opens the file if it is not already open. If the dBASE file is already open and has an associated index file, this file will become the current index file.

To declare this function, include the following statement in the linkDLL control structure:

```
INT selectDBFile(STRING)
```

Parameters

<file name>	the name of the file you want to make the current dBASE file and is any expression that evaluates to a valid dBASE file name.

Returns

1	if the function was successful
-3	if there are too many clients for this DLL, or not enough memory
-28	if there is no dBASE file with the specified name

If an error occurs, this function may return another negative number; use getDBErrorString to get an explanation of the error.

selectDBIndexFile

Function in TBKDB3.DLL

Syntax

```
selectDBIndexFile(<file name>)
```

Description

Makes the specified index file the current index file used to navigate in the current dBASE file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT selectDBIndexFile(STRING)
```

Parameters

`<file name>`	any expression that evaluates to the name of the index file to select.

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-53`	if there is no index file with the specified name

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

setCurrentDirectory

Function in TBKFILE.DLL

Syntax

```
setCurrentDirectory(<directory name>)
```

Description

Makes the specified directory the current working directory on a drive. Each drive maintains its own current directory.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT setCurrentDirectory(STRING)
```

Parameters

`<directory name>`	the name of the directory to make the current working directory.

Returns

`1`	the function was successful
`-3`	specified path name was invalid.

Examples

```
get setCurrentDirectory("c:\toolbook\training")
   -- Sets the current working directory to
   -- \TOOLBOOK\TRAINING.

get setCurrentDirectory(dirName)
   -- Sets the current working directory to the value
   -- of dirName
```

setCurrentDrive

Function in TBKFILE.DLL

Syntax

```
setCurrentDrive(<drive letter>)
```

Description

Makes the specified disk drive the current working drive.

To declare this function, include the following statement in the linkDLL control structure:

```
INT setCurrentDrive(STRING)
```

Parameters

<drive letter>	the letter designating the disk drive or any expression evaluating to a single letter.

Returns

1	if successful
-1	if there was an error

Examples

```
get setCurrentDrive("D")
   -- Makes drive D the current drive

get setCurrentDrive(driveLet)
   -- Makes the drive whose name is the value of the
   -- container driveLet the current drive
```

setDBDateFormat

Function in TBKDB3.DLL

Syntax

```
setDBDateFormat(<format string>)
```

Description

Sets the date format to be used in transactions through this DLL.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT setDBDateFormat(STRING)
```

Parameters

`<format string>`	any string that specifies a valid dBASE date format. The date format is specified as a "picture" of the date. The default date format is "MM/DD/YY" (month/day/year). Examples: YY.MM.DD CCYY.MM.DD MM/DD/YY DD-MM/YY DD-MM/CCYY MMM DD/YY

The "Y", "M" and "D" characters specify the year, month and day respectively. If there are more than 2 "M" characters, a character representation of the month will be used.

Returns

`1`	if the function was successful (even if the date format string itself was invalid)
`-3`	if there are too many clients for this DLL, or not enough memory

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

setDBFieldTag

Function in TBKDB3.DLL

Syntax

```
setDBFieldTag(<field tag number>, <field item>, <field name>,
   <field type>, <field width>, <field decimals>)
```

Description

Sets the specified field in the specified field tag to the specified field name, type, width, and decimal precision. You can use this function to specify this information for more than one field by calling it again for different fields with the same tag number.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT setDBFieldTag(WORD, WORD, STRING, STRING, WORD, WORD)
```

Parameters

<field tag number> — the number returned by the createDBFieldTag function.

<field item> — any expression that evaluates to number of the field to set. For details about setting the number of fields in a field tag, see the createDBFieldTag function.

<field name> — any expression that evaluates to the name of the field. The field name cannot be more than 10 characters long.

<field type> — any expression that evaluates to the field type which is one of the following strings:

"1" or "c" or "C"	(character)
"2" or "l" or "L"	(logical),
"3" or "d" or "D"	(date)
"4" or "n" or "N"	(numeric)
"5" or "m" or "N"	(memo).

These are dBASE field types. If the <field type> parameter evaluates to character, you must specify a field width with the <field width> parameter. If the <field type> parameter evaluates to date or memo, you do not need to specify a field width with the <field width> parameter. If the <field type> parameter evaluates to numeric, you must specify a field width with the <field width> parameter and you must specify the decimal precision with the <field decimals> parameter.

<field width> — an expression that evaluates to an integer indicating the width of the field. If this parameter is <=0 for a character field, then the <field width> parameter defaults to 254. If this parameter is <=0 for a numeric field, then the <field width> parameter defaults to 10. The maximum width for a numeric field is 19. If the field

type is memo, date or logical, this parameter is ignored.

<field decimals> — an expression that evaluates to the number of decimal places in a numeric field. It is ignored for other field types. If `<field decimals>` is 0 for a numeric field, then the contents of the field is treated as an integer. If it is more than 0, it must be smaller than, or equal to `<field width> - 2`. if `<field decimals>` is negative, larger than 15 or larger than `<field width> - 2`, the function will return an error.

Returns

`1` if the function was successful

`-6` if the field type is invalid

`-7` if the field tag is invalid

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
to handle newDBFile
   local dbFileName, noFields, fieldNames, fieldType
   local tagNumber, fieldWidth, i
   ask "What is the name of the new DB file?"
   put it into dbFileName
   ask "How many fields do you want in the new DB file?"
   put it into noFields
   ask "Type the field names you want separated by a comma"
   put it into fieldNames
      if itemCount(fieldNames) <> noFields
         do
            request "The number of field names and the" &&\
               "number of fields do not agree"
            ask "How many fields do you want for the" &&\
               "new DB file?"
            put it into noFields
            ask "Type the field names you want" &&\
               "separated by a comma"
            put it into fieldNames
         until itemCount(fieldNames) = noFields
      end if
      --iterate through each field name and ask for field
      --information
      step i from 1 to itemCount(fieldNames)
         put item i of fieldNames into it
         request "What is field type for field " & it & "?"
         put it && ", " into item i of fieldType
         put item i of fieldNames into it
         request "What is field width for field " & it & "?"
         put it && ", " into item i of fieldWidth
      end step
      set sysError to 1
      -- get return value for tag number
```

```
      put CreateDBFieldTag(noFields) into tagNumber
         if sysError = 0
            request "Error creating field tag"
            break newDBFile
         end if
      step i from 1 to itemCount(fieldNames)
         get setDBFieldTag(tagNumber,i,\
            item i of fieldNames, item i of fieldType,\
            item i of fieldWidth, 0)
         if sysError <= 0
            request "Error setting field tag"
            break newDBFile
         end if
      end step
      get createDBFile(dbFileName, itemCount(fieldNames), \
         tagNumber, 1)
         if sysError <= 0
            request "Error creating file"
            break newDBFile
         end if
      get freeDBFieldTag(tagNumber)
         if sysError <= 0
            request "Error freeing field tag"
            break newDBFile
         end if
end newDBFile
```

setDBFieldValue

Function in TBKDB3.DLL

Syntax

```
setDBFieldValue(<field name>, <new value>)
```

Description

Sets the contents of the specified field in the current record to the specified new value. Changes to field values made by this function are stored in the record buffer until you call the `writeDBRecord` function.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT setDBFieldValue(STRING, STRING)
```

Parameters

`<field name>`	any expression that evaluates to the name of the field whose value you want to change.
`<new value>`	any expression that evaluates to the new value for the specified field.

Returns

If no error occurs, the `setDBFieldValue` function returns `1`. If an error occurs, this function returns:

`-3`	if there are too many clients for this DLL, or not enough memory
`-4`	if the field name is invalid
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-26`	if a memo field could not be written
`-32`	no current record
`-60`	if trying to set a date field to an invalid date
`-62`	if the data is invalid for the field type
`-73`	if the text is too long to fit in the field

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
-- loop through pages of book and build dBASE records from
-- a ToolBook book containing record fields
step i from 1 to pagecount of book tb
   -- bypass pages not on the correct background
   if parent of page i of book tb is not bg
      continue step
```

```
   end if
   --loop through chosen recordFields of this background
   step j from 1 to fldct
      -- make local reference to data in recordField
      set src to "recordField" && textline j of \
         text of field "stbfn of page" && i && "of book" &&\
         quote & tb & quote
      set l to text of evaluate(src)
      -- check for truncation of data if occurs
      set k to textline j of text of field "swidth"
      if k is not null and charCount(l) > k
         set l to characters 1 to k of l
         increment totTrunc
      end if
      if setDBFieldValue(textline j of \
         text of field "sname", l) <> 1
         increment totInval
      end if
      get writeDBRecord(totRecs + 1)
      if it <> 1
         send error it, "writing dBASE record"
         break step
      end if
      increment totRecs
   end step
end step
```

setDBNavigateToDeleted

Function in TBKDB3.DLL

Syntax

```
setDBNavigateToDeleted(<option>)
```

Description

Sets a switch to allow navigation to records marked for deletion. The default is `off`.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT setDBNavigateToDeleted(INT)
```

Parameters

`<option>`	`0` to turn the navigation to deleted record off, or `1` to turn it on.

Returns

`1` if the function was successful

`-3` if there are too many clients for this DLL, or not enough memory

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

setDBRecordDeleted

Function in TBKDB3.DLL

Syntax

```
setDBRecordDeleted(<delete value>)
```

Description

Marks the current record for deletion.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT setDBRecordDeleted(WORD)
```

Parameters

`<delete value>`	determines if the current record is marked for deletion. If the value of the `<delete value>` parameter is greater than or equal to `1`, then the record will be marked for deletion. If the value is less than or equal to `0`, then this function will remove the record's mark that indicates it is to be deleted.

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-12`	if there is no current dBASE file; you must use `openDBFile` first
`-32`	if there is no current record; you must use a function such as `firstDBRecord` first

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

setFileAttributes

Function in TBKFILE.DLL

Syntax

```
setFileAttributes(<file name>, <attributes>)
```

Description

Set the file attributes for the specified file.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT setFileAttributes(STRING, STRING)
```

Parameters

`<file name>`	the name of the file whose attributes you want to set.
`<attributes>`	a string containing one letter for each of the attributes you want to set: `R` to set the Read only attribute to true `H` to set the Hidden attribute to true `S` to set the System attribute to true `A` to set the Archive attribute to true

Returns

`1`	if the function was successful
`-2`	if the specified file was not found
`-3`	if the specified path name was invalid
`-5`	if access to the file was denied.

Examples

```
get setFileAttributes("c:\TBKHelp\help1.tbk", "RHA")
   -- Sets the attributes for the file HELP1.TBK to
   -- read only, hidden, and archive

put setFileAttributes(fileName, fileAttr) into the commandWindow
   -- Sets the attributes of the file whose name is
   -- the value of fileName to the attributes
   -- represented by the value of fileAttr.

get setFileAttributes("c:\config.sys", null)
   -- Sets all four attributes of config.sys to false
```

setWinIniVar

Function in TBKWIN.DLL

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.

To declare this function, include the following statement in the `linkDLL` control structure:

```
WORD setWinIniVar(STRING, STRING, STRING)
```

Parameters

`<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 WIN.INI.
`<item name>`	any expression that evaluates to the name of the item whose value you want to set.
`<new value>`	any expression that evaluates to the new value you want to set for the specified item.

Returns

If no error occurs, the `setWinIniVar` function returns `1`.

Example

```
if setWinIniVar("GameBook","GreetingMessage","false")<1 then
   request \
   "Oops! Won't remember to turn off the greeting message!"
end if
```

verticalDisplayRes

Function in TBKWIN.DLL

Syntax

```
verticalDisplayRes()
```

Description

Returns the physical display's vertical size in pixels.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT verticalDisplayRes()
```

Parameters

None

Returns

If no error occurs, the `verticalDisplayRes` function returns the vertical display size in pixels.

Example

```
put "Screen dimensions in pixels:" &&\
   horizontalDisplayRes() & "," & verticalDisplayRes() \
   into commandWindow
```

verticalDisplaySize

Function in TBKWIN.DLL

Syntax

```
verticalDisplaySize()
```

Description

Returns the physical display's vertical size in millimeters.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT verticalDisplaySize()
```

Parameters

None

Returns

If no error occurs, the `verticalDisplaySize` function returns true.

writeDBRecord

Function in TBKDB3.DLL

Syntax

```
writeDBRecord(<record number>)
```

Description

Writes the contents of the record buffer into the specified record and updates all open index files. The contents of the record buffer is set with the `setDBFieldValue` function.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT writeDBRecord(DWORD)
```

Parameters

`<record number>`	the number of the record to which you want to write the contents of the record buffer. If the value of the parameter is `0`, then the current record is updated with the contents of the record buffer. If the `<record number>` parameter indicates a higher record number than is in the dBASE file, ToolBook will append a new record that contains the contents of the record buffer. If the `<record number>` parameter specifies a record with contents, the `writeDBRecord` function will overwrite the contents of that record.

Returns

`1`	if the function was successful
`-3`	if there are too many clients for this DLL, or not enough memory
`-8`	if the file, the index or the system got corrupted
`-9`	if the function failed because of a duplicate key
`-12`	if there is no current dBASE file
`-79`	if the record was marked as deleted, write operation was denied.

If an error occurs, this function may return another negative number; use `getDBErrorString` to get an explanation of the error.

Example

```
-- loop through pages of book and build dBASE records from
-- a ToolBook book containing record fields
step i from 1 to pagecount of book tb
   -- bypass pages not on the correct background
   if parent of page i of book tb is not bg
      continue step
   end if
   --loop through chosen recordFields of this background
```

```
   step j from 1 to fldct
      -- make local reference to data in recordField
      set src to "recordField" && textline j of \
         text of field "stbfn of page" && i && "of book" &&\
         quote & tb & quote
      set l to text of evaluate(src)
      -- check for truncation of data if occurs
      set k to textline j of text of field "swidth"
      if k is not null and charCount(l) > k
         set l to characters 1 to k of l
         increment totTrunc
      end if
      if setDBFieldValue(textline j of \
         text of field "sname", l) <> 1
         increment totInval
      end if
      get writeDBRecord(totRecs + 1)
      if it <> 1
         send error it, "writing dBASE record"
         break step
      end if
      increment totRecs
   end step
end step
```

xPixelsFromUnits

Function in TBKWIN.DLL

Syntax

```
xPixelsFromUnits(<xdimension>)
```

Description

Converts a horizontal dimension in ToolBook page units into a dimension in screen pixels. This function can be useful because the number of page units per pixel may vary with the display device.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT xPixelsFromUnits(INT)
```

Parameters

`<xdimension>`	an integer number or an expression that evaluates to an integer number in the range `-32768` to `32767`.

Returns

This function returns a number of pixels.

xUnitsFromPixels

Function in TBKWIN.DLL

Syntax

```
xUnitsFromPixels(<xdimension>)
```

Description

Converts a horizontal dimension in screen pixels into a dimension in ToolBook page units. This function can be useful because the number of page units per pixel may vary with the display device.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT xUnitsFromPixels(INT)
```

Parameters

`<xdimension>`	an integer number or an expression that evaluates to an integer number in the range `-32768` to `32767`.

Returns

This function returns a number of ToolBook page units.

yieldApp

Function in TBKWIN.DLL

Syntax

```
yieldApp()
```

Description

Halts the current task and starts any waiting task.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT yieldApp()
```

Parameters

None

Returns

If no error occurs, the `yieldApp` function returns a non-zero integer.

Example

```
-- "well-behaved" long pause
while sysTime - startTime <  1000
   set dummyVar to yieldApp()
end while
```

yPixelsFromUnits

Function in TBKWIN.DLL

Syntax

```
yPixelsFromUnits(<ydimension>)
```

Description

Converts a vertical dimension in ToolBook page units into a dimension in screen pixels. This function can be useful because the number of page units per pixel may vary with the display device.

To declare this function, include the following statement in the linkDLL control structure:

```
INT yPixelsFromUnits(INT)
```

Parameters

<ydimension>	an integer number or an expression that evaluates to an integer number in the range -32768 to 32767.

Returns

This function returns a number of pixels.

yUnitsFromPixels

Function in TBKWIN.DLL

Syntax

```
yUnitsFromPixels(<ydimension>)
```

Description

Converts a vertical dimension in screen pixels into a dimension in ToolBook page units. This function can be useful because the number of page units per pixel may vary with the display device.

To declare this function, include the following statement in the `linkDLL` control structure:

```
INT yUnitsFromPixels(INT)
```

Parameters

<ydimension>	an integer number or an expression that evaluates to an integer number in the range `-32768` to `32767`.

Returns

This function returns a number of ToolBook page units.