OKScript 4.0 Help Contents

Thank you for using OKScript - a shareware scripting tool for applications that run under Microsoft Windows 3.1 or later, available in both 16 bit (for Windows 3.x) and 32 bit versions.

Although OKScript is a general purpose scripting tool, much of this documentation is oriented toward OKbridge players (see www.okbridge.com), the original target audience for the program. If you are a new OKScript user and intend to use if for OKBridge this topic may be all you need to get started, otherwise the introduction and tutorial would be good places to start.

OKScript 4.0: Easy Scripting, Button Panels and More

OKScript provides a simple and flexible way to send keystrokes to any application designed for Microsoft Windows (3.1 or later.) Basic scripts can be written in seconds (often requiring only one line of easy to learn code) with the built-in multi-file script editor. Once written, these instructions are carried out by clicking a button or menu item in a compact (optionally rollup) panel that OKScript produces from the script, or by one of OKScript's advanced activation methods.

OKScript is ideal for automating awkward or repetitive keyboard tasks, particularly for those applications that don't have built-in scripting or macro facilities. Although originally designed to ease the typing burden in OKbridge, an online contract bridge club (see www.okbridge.com), OKScript has evolved into a widely applicable general purpose scripting tool. Often an OKScript solution is preferable to what can be implemented in the target application's native scripting and/or macro tools. OKScript has also been useful in moving data between applications with incompatable file formats by automating clipboard transfers.

Users of OKScript version 2.x will find many ease of use improvements in the latest release. Instructions for converting older scripts can be found here.

The OKScript Modes

OKScript is always operating in one of two modes. In Edit Mode OKScript displays a multi-file script editor. Filenames are displayed on tabs below the editing area. (Files that have the .OKS extension are displayed in the tabs without extension to save tab space. Files without any extension are disambiguated with a colon (:) suffix.) You can switch from file to file by simply clicking on the desired file's tab. If there is insufficient space for all your tabs to be displayed you can sequence through them by clicking on the arrows in the lower right corner of the OKScript window.

In Run Mode the editor disappears and OKScript attempts to build the button and/or menu panel defined by the script currently visible in the editor. If something goes wrong you are informed with an error message and returned to the editor with the cursor on the flawed line. Once your scripts run successfully you can switch from file to file by clicking on the still visible tabs, or sequence through them with ctrl-tab and shift-ctrl-tab. This feature allows you to have separate scripts for many different applications instantly available in just one copy of OKScript.

You can switch between Edit and Run modes through the File menu or by hitting the ctrl-E and ctrl-R keys.

Simple Scripting for OKbridge

Useful one-line scripts really *can* be written and up and running in seconds! Here is a script for OKbridge that produces a button that alerts a Precision 1C opening:

```
BUTTON 1C red OKbridge =16{+} points, any distribution{enter}
```

For pickup partnerships, OKScript's plagiarize feature and a palette of pre-built scripts lets you build custom panels in seconds by simply clicking on the buttons that you want from your palette. This is a great way to quickly document your agreements. Visit the web site's samples page for a more elegant approach to this problem.

For more information on building scripts visit the scripting tutorial. OKScript's language reference section contains complete information on all of OKScript's scripting instructions.

Contacting the Author

OKScript is still a growing product. If you find bugs, or a feature that would be particularly helpful, please let me know, at:

yweare@gte.net

and include the OKScript version number, 'bitness' (16 or 32), the operating system of your machine (Windows 3.1 or 95 or NT) and any script code that you are having trouble with. Should this email address fail, check the OKScript website for current support information:

http://home1.gte.net/yweare

License Agreement and Disclaimer

End User License Agreement
This notice must accompany any distribution of this software. This document supersedes all previous distribution policies.

This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and the author of the OKScript software product which includes the computer software, documentation and associated media, which is termed the "SOFTWARE PRODUCT". By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE PRODUCT.

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is owned by the above mentioned author and is licensed, not sold.

As an *unregistered* end user you are hereby granted the rights to use the SOFTWARE PRODUCT for non-comercial applications. Except for evaluation purposes, unregistered comercial use is prohibited.

As a *registered* end user you are granted the rights to use the SOFTWARE PRODUCT for non-comercial *and* comercial applications. Registering the software and installing the license releases functionality limits that are present when OKScript doesn't detect a valid license. (See below.)

The SOFTWARE PRODUCT shall not be resold, leased or rented, including, but not limited to, distributing OKScript as part of commercial products, or in support of commercial services, without the expressed written authorization of the author. Under no circumstances shall non-expiring OKScript license codes be published or disclosed through any medium.

You may personally give away copies of the program, but not license codes, through private channels, such as e-mail, provided that the program and the distribution package are not renamed or modified. The original standard distribution package contains six files: okscript.exe, oks.hlp, setup.oks, setup.txt, palette.oks and examples.oks. Some distributions may contain additional sample (.oks and .txt) files. No distribution shall contain the file okscript.ini. Except for authorized OKScript affiliates you may not redistribute this software via a web or ftp site. You may (and are encouraged to) provide links to an authorized distribution site, such as http://home1.gte.net/yweare.

Teachers and educational institutions may distribute unlicensed copies of the program to students for free or for minimal copying costs if the software is to be used in a course.

You are prohibited from modifying or reverse engineering this software or any of its components.

Licensing
Your copy of OKScript becomes licensed when you register it and enter the license information provided. The current supplier of OKScript registration services is posted at: http://home1.gte.net/yweare. Upon registration you will receive information that is to be entered

into OKScript's registration form which is accessed from the **Help | Register** menu. By licensing OKScript you will:

-No longer see the opening mode selection dialog.
-Be able to run scripts without the size or function limitations of the unlicensed version.
-Be free to use OKScript commercially.
-Receive OKScript news, update notices and free technical support.
-Be licensed for downloaded updates, as they become available, for two years.
-Be supporting the continued development of OKScript.

Licenses are for single users of the SOFTWARE PRODUCT. You are prohibited from having a license installed on more than one machine at a time, except in situations where it is impossible for both copies to be running concurrently, for example, a business machine and a home machine. You are also prohibited from any communication or publication of the license code information.

Short term temporary licenses are available for OKScript classes and seminars. Contact the author (see below) for more information.

Disclaimer

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN PARTICULAR, THIS SOFTWARE IS NOT FAULT-TOLERANT AND IS NOT DESIGNED OR INTENDED FOR USE IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE. ACCORDINGLY, THE AUTHOR DISCLAIMS ANY LIABILITY FOR ANY CLAIMS OR DAMAGES ARISING FROM THE USE OF THIS SOFTWARE IN SUCH APPLICATIONS.

Reporting Bugs and Getting Help

Email problem reports to the author at **yweare@gte.net**. Please include any source code and procedures needed to reproduce the problem, the OKScript version number (in **Help | About** box) and 16 or 32 bit-ness, and any other information that might help me locate and correct the cause. Check http://home1.gte.net/yweare for possible alternative support email addresses.

The OKScript Command Line

OKScript can be started from the Program Manager, a Win95/98/NT shortcut (icon) or a RUN dialog. This topic describes only the command line required to launch OKScript. Click here for instructions on setting up OKScript icons on a 32 bit system, or here for setting up a Program Manager link to OKScript for 16 bit systems. (This information is also in the Setup.txt file of the 16 bit version.)

Two options can be specified on the OKScript command line:

```
path\OKScript.exe [/ntitle] [/e] [file1 file2 ...]
```

/ntitle causes OKScript to be launched with *title* as the caption in the title bar. For the 32 bit version of OKScript titles with spaces may be put in quotes, thus /N"OK Script". The 16 bit version must have titles containing no spaces. This feature is useful when you want two OKScript panels to interact with eachother, in which case they will need different activation names.

/e in the command line causes OKScript to start in editor mode, as does a command line containing no script files. If one or more script files are specified then they all will be loaded into editor pages and if there is no ***/e*** present then a panel will be built from the last file specified. (Note that this switch replaces the functionality provided by the ***/r*** switch in previous versions.)

The remaining command line argument(s) specify the script files to be loaded. These names can be either .oks files or directory names. When a directory name is used all the .oks files in that directory are loaded. Here are two typical command lines:

```
c:\oks40\okscript.exe raises.oks defense.oks system.oks
c:\oks40\okscript.exe /e /t /nOKS2 c:\oks40\mydir
```

Note, OKScript enters "setup" mode if it is started with no file arguments or switches, and it finds a file called setup.oks in the current directory. The following events take place in this setup mode:

- an option to generate a shortcut is offered (32 bit version only.)
- setup.oks is loaded and executed (a button panel usually appears.)
- setup.oks is then deleted, ensuring that this mode will only be executed once.

Except as a part of the OKScript installation sequence it is unlikely that you would find any use for this feature, but be forewarned that any script named SETUP.OKS is vulnerable to deletion by OKScript.

OKScript Tutorial

This tutorial will step you through the writing of a few basic scripts. In this writeup the vertical bar notation, e.g. **File | Open**, refers to a specific menu item, in this case it is an instruction to click the **Open** item in the **File** menu.

Begin by starting the Windows Notepad application and drag it out of the way (but don't minimize it.) (Move a window by putting the mouse pointer on the title bar at the top of a window, holding down the mouse button and drag the window where you want it, then release the button.) We will get back to the Notepad in a moment.

Next - one click scripting

Converting Scripts from Older OKScript Versions

From version 2.x

Version 3.x and 4.x of OKScript represent a significant departure from the widely used 2.x, and primarily targets the ease of use concerns that a year's experience and perspective uncovered in the earlier implementations. Current users of 2.x will find the following changes and enhancements:

- An integrated, multi-file script editor, which...
- Reports the location of script errors.
- Built-in help.
- Run-time macro substitution.
- 50 percent reduction in the number of instructions.
- Elimination of data manipulation.
- Higher level control flow functionality.
- Consolodation of button declaration functions.
- Screen position memory.
- Runtime panel switching through the editor tabs.
- 'Point and shoot' quick button panel creation.
- Secondary button scripts activated with right mouse button.
- Create Windows 9x desktop shortcut from the OKScript menu.
- Menu activated scripts.
- File and directory manipulation instructions.

For most pre-version 3.x scripts there is a simple correspondence to the new instruction structure. Consider this typical 2.x script:

```
BUTTON 1club
COLOR red
ACTFUZZ OKbridge
SENDXI 16{+} points, any distribution{enter}
END
```

This is converted to the new style by simply removing all the instruction names (except BUTTON) and line breaks, yielding:

```
BUTTON 1club red OKbridge 16{+} points, any distribution{enter}
```

A script that performs this conversion is left as an exercize for the reader.

From Version 3.x

Generally there will be little required when upgrading from version 3.x to 4.x. The DATE and TIME actions have been removed and replaced with the -DATE and -TIME macros. So, the DATE instruction becomes PUT %-DATE% and TIME becomes PUT %-TIME%. The CLIPTOMAC function has been removed, replace it with LET mac %-clipboard%. Finally, the IF instruction's syntax has been changed a bit.

File Menu Reference

new - open - save - run/edit - plagiarize - fwd/back - make icon - exit

The File Menu provides access to all of OKScript's file and program management functions. Some of the entries in this menu are only available in Edit or Run mode, but not both.

New (Ctrl+N)
Creates a new, empty, editor page without a name. Returns to edit mode if activated from run mode.

Open (Ctrl+O)
Creates a new editor page and Opens a "File Open" dialog box allowing the user to select a file to read into the editor. Returns to edit mode if activated from run mode. The 16 bit version of OKScript has a file size limit of about 32k.

Open and Go (Ctrl-G)
The same as the Open directive except OKScript will attempt to process the file into a button panel after it is read into the editor. The 16 bit version of OKScript has a file size limit of about 32k.

Save (Ctrl-S)
Saves the currently active editor text. The user is prompted for a file name if the current editor page hasn't yet been assigned one yet.

Save As
Prompts the user to select a file into which the currently active editor text will be saved. If the filename has no extension, or is other than ".oks" then ".oks" will be added, unless the entire filename is enclosed in "quotes", e.g. "myfile".

Run Panel (Ctrl-R)
Available only in edit mode, Run assembles the contents of the current editor page into a button panel.

Edit Panel (Ctrl-E)
Not available in Edit mode, this control activates the editor on the page containing the script for the current button panel.

Plagiarize | To This Page (Ctrl-T)
Causes OKScript to enter Plagiarize mode with the current editor page as the target (destination.) In this mode buttons and user defined menus that you click on are not executed. Instead, their script code is appended to the target editor page. This process continues until either a) File | Edit-Panel is hit, b) the tab of the target page is hit, or c) File | Plagiarize | Stop is hit. Multiple BUTTON and MENU declarations, and supporting PROCs, can be plagiarized in one click by using the PLAG and GALP directives.

This mode is particularly useful for creating quick button bars for documenting, for example, pickup partnership agreements. If all the conventions that you play are scripted in a convention library which is loaded into OKScript you can simply plagiarize the buttons that you want into a new panel. A NEWLINE directive can be inserted into the plagiarized script by hitting the **Newline!** menu item, which is only visible when plagiarizing.

Palette.oks is a sample script library included with the OKScript distribution. This file is only

intended as a guide and may be modified in any way that you wish to suit your particular needs.

Plagiarize | To New Page (Ctrl-P)
Works just like Plagiarize | To This Page, described above, except creates a new editor page and makes that the target.

Plagiarize | Stop (Ctrl-Q)
Stops Plagiarize mode.

Page Fwd (Ctrl-Tab)
Sequences forward through the scripts in the script editor. If performed while a button panel is active (i.e. not editing) then the new script page will be run, creating a new button panel.

Page Back (Shift-Ctrl-Tab)
Sequences backward through the scripts in the script editor. If performed while a button panel is active (i.e. not editing) then the new script page will be run, creating a new button panel.

Make Shortcut
Builds a shortcut (icon) that will start OKScript in the current configuration. The dialog box asks for the name to display with the shortcut, the title bar caption (OKScript is the default), the default folder for script files, and where the shortcut should be placed - the desktop or within the Start Menu structure. This function is available only in the 32 bit version of OKScript.

Exit
Exit terminates OKScript after prompting the user to save any modified editor pages.

Edit Menu Reference

undo - cut/paste - close - search/replace - bookmark - key record - color - font

The Edit Menu provides access to all of OKScript's script editing functions. The Edit Menu not available in run mode.

Undo (ctrl-Z)
Reverses the last 'reversable' editing operation performed. Undo is not enabled during keystroke recording.

Cut, Copy, Paste and Select All
The clipboard functions operate just like all Windows applications, allowing the user to transfer text from one editor page to another, and between applications.

Close Page
Closes the currently active editor page.

Close All
Closes all the open editor pages.

Search (and Replace) and Repeat Search (F2 or ctrl-F and F3)
Edit | Search (**F2**) allows you to search for (and optionally replace) text in the current editor page. Enter the desired search text in the dialog box, and check the case sensitive search option if you want it to be a case sensitive search. Check the search and replace box and fill in the replace string if you wish to do that operation. Hit the OK button to perform the search starting at the current cursor location. (Note that if any text is selected the search will begin at the *end* of the selection, regardless of which end the cursor is at.)

To repeat the search hit **F3** or click on **Edit | Repeat Search**. Note that you can reject a replace operation (restoring the original text) by hitting the **ctrl-Z** (**Edit | Undo**) key.

Bookmark (F9-12 and Ctrl-F9-12)
The **Edit | Bookmark | Set** menu remembers the script editor's current cursor position. Use Ctrl-F9, -F10, -F11 and -F12 to quickly set a bookmark. Each editor has it's own set of four bookmarks. **Edit | Bookmark | Go** returns the cursor to the position previously saved in the bookmark. Use F9 - F12 to go there in one keystroke. Bookmarks are adjusted when text is inserted or removed.

Record and Playback Keys (F4-F8)
The OKScript editor allows you to record a series of keystrokes which can be later played back. To start recording hit **Edit | Record Keys** (or **F4**), then type the desired keys. To save these keypresses hit **F5**, **F6**, **F7** or **F8**. (To cancel the recording hit **F4** again, or **Edit | Record Cancel**.) Once saved the keypresses can be played back just as they were recorded by hitting the F-key they were saved with. Note that the only menu functions that are recorded are **Search** and **Repeat Search**. The keyboard recording capability is useful for performing repetitive editing tasks including search and modify operations that are too complex for the built-in search and replace (discussed above). All but the longest stored keypress strings are saved at the end of an OKScript session and available in future sessions.

White on Black
This control toggles the editor color from white text on black to black text on gray (or the Windows default background.) The change is recorded and will be active the next time you use

OKScript.

Font
This control allows you to select the editor font. The change is recorded and will be active the next time you use OKScript.

Iteration

loop - break - Notation and Layout

OKScript provides a minimal set of control flow instructions to facilitate advanced application scripting. Procedure (subroutine) and conditional (IF THEN ELSE) capabilities are also provided and discribed in separate help topics.

LOOP *count* ... *instructions* ... POOL

LOOP causes the following *actions*, up through the POOL instruction, to be executed *count* times. *Count* may be a macro reference. A LOOP may be terminated prematurely with the BREAK instruction (see below.) The maximum nesting level for LOOPs is 10. Here is an example of a LOOP:

```
LOOP 5
  PUT sometext
  POOL
```

BREAK

Causes the immediate termination of a loop. Execution resumes immediately after the POOL instruction.

```
LOOP 5
  CALL someproc
  IF TEXT %-clipboard% EQ abc
    BREAK
    FI
  POOL
```

OKScript Menu Reference

This menu documentation uses the vertical bar notation, e.g. **File | Plagiarize | Stop**, refers to a specific menu item, in this case 'Stop' item in the 'Plagiarize' submenu of the 'File' menu.

Many of the menu entries have shortcut keys which activate the function directly without the need to enter the menu structure.

FAQ - Frequently Asked Questions

How do I ...

Other Questions ...

OKScript Revision History

1.0 10/96

1 initial release.

2.0 12/96

1 Added clipboard and data manipulation instructions.

2.1 10/97

1 Fixed initial 'flash' in 32bit version.
2 Added keyboard navigation capabilities.
3 Fixed LOADSCRIPT functionality.
4 Refined code that tracks active windows.
5 Included sample script file.
6 Changed to Mijenix's ZipMagic(r) self extracting installation shell.

3.0 12/97

1 Added integrated script editor.
2 Changed scripting language.
3 Added online help

3.1 12/97

1 Added code to remove trailing blanks when a file is read or processed.
2 Added feature to allow user to remember window position in a script.
3 Allowed user to switch button panels via the editor tabs.
4 Added script building from button palette features.

3.1a-e 12/97

1 Minor bug fixes and interface enhancements.

3.2 12/97

1 Added secondary button scripts via right mouse button.
2 Improved plagiarize mode access.
3 Fixed sporadic synchronization problem that occured when a RUN instruction was followed by a WINDOW instruction.

3.2b 1/98

1 Added OPEN and CLOSE instructions.
2 Added macro persistence.
3 Fixed 'stay on top' problem with GET instruction.
4 Fixed clicking on unpopulated part of form fault.

3.3a 1/98

1 Added control flow instructions.
2 Fixed failure to plagiarize on new instructions.

3.3b 1/98

1 Fixed horrible flaw that caused okscript to lock up.

3.3c 1/98

1 Improved CALL instruction performance.
2 Forced STARTUP to be action execution terminator.
3 Fixed PROC implementation bugs.

3.3d 1/98

1 Fixed bug that made program 'lose' help file.

3.3e 1/98

1 Added Beautify function to the Edit menu.

3.3f 1/98

1 Added nesting check and else with condition capability.

3.3g 1/98

1 Fixed bug affecting the interaction of the STARTUP, OPEN and CLOSE instructions at compile time.

3.3h 2/98

1 Fixed window flash when exiting OKScript with multiple files open.

3.3i 3/98

1 Fixed syntax check and beautify code so it properly recognizes blank lines.

3.3j 5/98

1 Fixed implementation of tab display parameter in PLACE directive.
2 Added /T command line switch to inhibit the display of runtime tabs.
3 Improved editor to notebook tab coupling and button control implementation.

3.3k 5/98

1 Added {%} and {QC}.
2 Forced GET dialog edit control to always be active.
3 Properly restore buttons from canceled program exit.

3.3l 5/98

1 Corrected bug (introduced in 3.3k) that inhibited macro expansion in window names.

3.4a 6/98

1 Added editor search and keystroke record/playback.
2 Added TOPANEL instruction.
3 Added /N switch to set the title bar caption.
4 Replaced /R switch functionality with /E switch.
5 Imporved help file.
6 Added desktop shortcut menu function (32 bit only).

3.4b 7/98

1 Added macro expansion to the BUTTON directive's color parameter and the parameter of the CLIPTOMAC and CALL instructions.
2 Removed parameter quoting in WINDOW, OPEN and TOPANEL instructions.
3 Fixed directory binding error for file operations.
4 Improved help structure.

3.5 7/98

1 Added one level of editor undo.
2 Added replace option to text search.
3 Added MENU directive.
4 Added optional 'next panel' parameter to the CLOSE action.
5 Inihibited OPEN instruction from loading same file more than once.

4.0 9/98

1 Reorganized menus.
2 Added Newline menu for plagiarizing and removed special NEWLINE palette button.
3 Added Abort_Script menu.
4 Added DO, PLAG and PRAGMA directives.
5 Added LET, TRUNC, PARSE, FINISH, ONERROR, FILE, DIR, STATE, MESSAGE, MODE and WAITFOR actions.
6 Added QUERY, RUNNING, EXISTS, MENU and BUTTON iftypes.
7 Added parameter facility to PROC/CALL.
8 Added USE directive for separate library files.
9 Added integer formulas.
10 Added rollup argument to PLACE directive.
11 Added ability to load all .OKS files in a directory.
12 Added error recovery to TOPANEL.
13 Added support for diacritic characters.
14 Added multi-line instruction support (MORE).
15 Added pulldown option list to GET instruction.
16 Added panel and editor status bars.
17 Added editor bookmarks.
18 Added ability to insert delays after each keystroke or enter sent.
19 Eliminated need to click on a window for OKScript to 'see' it.
20 Extended OPEN instruction to allow file replacement and directory load.
21 Moved Newline menu item to File menu.
22 Moved trailing blank stripping to menu.
23 Changed undo from alt-BkSp to Ctrl-Z.
24 Increased the maximum file size in 32 bit version.
25 Improved button right click look and feel.
26 Changed IF instruction syntax.
27 Moved the /m, /s, /t, /c and /f switch functions into the OKScript menus and .INI file.
28 Removed ".OKS" from editor tabs, other extensions continue to be displayed.
29 Removed TIME, DATE and CLIPTOMAC actions.
30 Predefined macros: -date, -time, -clipboard, -tic, -ticwin, -actwin, -titles, -curdir, -args, -argn, -indexup, -indexdn, -found, -exe, -panel, -cancel, -myname, -mycolor, -mycheck, -err and -res.
31 Improved text search and beautify performance.
32 Fixed plagiarizing and CapsLock anomalies.
33 Fixed user menu related memory leak.
34 Fixed fast keystroke playback problem in editor.
35 Fixed bugs in beautify.
36 Metacharacters ignored on all but key sending instructions, but now they retain their meta function within macro expansions.

"Can't find window" error

This error happens when an OKScript button or menu is clicked and OKScript can't find the

target window that the script wants activated. OKScript returns to the editor with the cursor on the faulty instruction. This problem is usually caused by an error in the case of the script's window specification, e.g. OKBridge instead or OKbridge, or the window specification is misspelled. Correcting the script should solve the problem.

Moving a shortcut to the Start Menu

It may be easier to simply build a new Start Menu shortcut with the **File | Make Shortcut** capability. Otherwise, the first step is to click and drag the OKScript icon to the start button and drop it there. This will add the OKScript icon to the part of the start menu that appears when you first click on it. If you want it at a lower level of the start menu do the following:

- Right click (i.e. with the right mouse button) the start button and select OPEN from the menu that appears. This will give you a window view of the modifiable part of your start menu.
- Double click on the icon representing the submenu that you want to receive the OKScript icon. This will open up another window.
- Continue to "drill down" into your menu structure until you get to the place that you want OKScript to appear.
- Drag the OKScript icon (from the initial start menu window) to this window (moving around the windows so that you can see both of course.)
- Close all these windows by clicking the 'X' in the upper right corner.

A final note, the Make Shortcut menu function can add shortcuts to any folder in the Start Menu.

Why can't I repeat a lobby call?

The OKWin client program for OKBridge sometimes seems to filter out messages that are identical to the preceeding message. Try alternating between two similar messages.

How do I just build a simple message button?

Assuming that you want the message to go to the OKWin client for OKBridge the following line (and more like it) can be entered in the editor (hit **File|Edit Panel** if not already in the editor) and compiled by hitting **File|Run Panel**:

BUTTON welcome red OKbridge Welcome everyone, good luck{enter}

This line makes a button that reads "welcome" in red and sends "Welcome everyone, good luck" to the window with "OKbridge" in its title bar. The "{enter}" sends the keyboard 'enter' key just like you would do if you were typing it manually into OKWin.

Note that OKbridge must be spelled exactly as shown. This field in the button line is sensitive to the case of the characters.

Why are some characters ignored in messages?

The caret (^), plus (+) and tilde (~) characters are OKScript metacharacters. They have special meanings in OKScript text that is to be sent as keypresses. To override these special meanings, so as to be able to send these characters to other applications, put the metacharacter in braces, e.g.: {+}.

This also applies to the current macro quote character, which is the percent (%) character by default, but can be changed with the QUOTECHAR directive. The percent character is used in OKbridge to send messages to the spectators and is sent by putting it in braces. The current quote character can be sent by including {QC} in the message.

Switching to Another Notebook Tab

Some applications have what looks like a multi-section notebook with clickable tabs to change sections. (The OKScript editor is one such application.) The user can usually switch between pages of these notebooks with some key sequence, often ctrl-tab and ctrl-shift-tab. These codes can be included in an OKScript message, something like ^{TAB}.

In a few applications, notably Microsoft Exchange property windows, this doesn't seem to work and I haven't found a work around at the time of this writing.

Summary of Scripting Instruction Codes

directives - actions - control flow - Notation and Layout

Directives

Directives perform *organizational* tasks in OKScript - defining button, panel and symbol characteristics. **Note that a directive that can have multiple lines is terminated when any of these directives are encountered.** Click here to see how to use the MORE instruction to break long directive lines into multiple lines.

MACRO *name text*
QUOTECHAR character
BUTTON *"name" [width] color ["window" [text]]*
RIGHT *["window" [text]]*
MENU *"menu path" ["window" [text]]*
NEWLINE
PLAG and GALP (PLAG is not a block terminator.)
PLACE *left top runwithtabsflag rollupflag*
STARTUP
PROC *procname*
USE *tabname*
DO <AT | EVERY | IN> *[date] time*
DO <BEFORE | AFTER> <MENU | LEFT | RIGHT> *name*
PRAGMA <MENUSTUBS | MENUPROPER | STATUSBAR | FILEHELPOFF | WINERROROFF>

Actions

Action instructions perform their function when a script is executed, that is, when a scripted button or menu is pressed, the startup section of a script is encountered, a DO event occurs or a PROC is called. They have nothing to do with the actual construction of the button panel, but everything to do with what the panel's buttons (and/or menus) do. (Ordered by expected frequency of use.)

MORE *text*
WINDOW *windowname*
GET *macroname prompt*
PUT *text*
DECODE *encodedtext*
LET *[<* | #>]macroname text*
TOCLIP *text*
MESSAGE *text*
DELAY *x*
RUN *"dir" "program" parameters*
FILE *op parameter(s)* (All but the last parameter may be quoted.)
DIR *op parameter*
ONERROR *flagvalue*
OPEN *[*]scriptfile*
CLOSE *[next]*
TOPANEL *tabname*
MODE <ON | OFF | FLIP | TITLE | STATUS> [<BUTTON | MENU | MENUCHECK | *color*>] <*"name"* | *text*>
STATE *op statename*
TRUNC *macroname count*
PARSE *"delimiter" destmacro sourcemacro*
STOP

Control flow actions

Control flow actions affect the sequence in which script actions are performed.

IF *iftype arg predicate text ... actions* [ELSE ... *actions*] FI
LOOP *count ... actions ...* POOL
BREAK
WAITFOR *windowname*
CALL *procname [arguments]*
FINISH *[message]*

Panel Initialization

Notation and Layout

PLACE *left top runwithtabsflag rollupflag*

Creates the OKScript panel at the screen position specified by *left* and *top*. If *runwithtabsflag* is greater than zero then the tabs are shown on the button panel; if zero then they are hidden; if negative then the current tab display mode is used. A default setting can be set in **Script | Set Panel Options**.

If *rollupflag* is positive then rollup is enabled; if zero then disabled and if negative the default rollup mode is used. When rollup is enabled OKScript "rolls up" into just a window title bar when another application is active and "unrolls" when it becomes active, e.g. by clicking on it. A default rollup value can be set in the **Script | Set Panel Options**.

STARTUP

Code that follows this directive is executed (until a terminating directive is encountered) immediately after the script file is compiled into a panel (run.)

USE *tabname*

OKScript allows the PROC, MACRO, MENU, BUTTON, NEWLINE, PLACE, DO and PRAGMA and additional USE directives to be placed in files that are separate from the referencing script. To access these *external* directives insert a USE instruction in the script that references them. USE allows you to build reusable macro, procedure, button and menu libraries. Such a library may also reference other libraries by including its own USE instructions. Library files must be loaded into OKScript's editor before they are USEd. USE's argument is the name displayed on the library's editor tab.

Defining Panel Buttons and Menus

button - right - newline - menu - Notation and Layout

BUTTON *"buttonname" [width] textcolor* [*"activewindow"* [*texttosend*]]

Creates a script button that is *width* pixels wide, named *buttonname* displayed in *textcolor.* If width isn't present the button is autosized. Pressing the button that is created by this script will activate the window containing *activewindow* (if present) in it's title bar and send the string *texttosend* (if present) to that window. If *activewindow* contains spaces it must be "quoted". *Texttosend* may contain OKScript metacharacters. Actions, including control flow instructions, following the button directive will also execute when the button is pressed, up until the next terminating directive.

Note that an ampersand (&) in the buttonname field causes the letter following the ampersand to be shown underlined. This character acts as a hotkey for this button - you can activate that button by hitting alt-x, where x is the underlined letter. Use && in the name field to override this feature and actually display an ampersand on the button.

RIGHT [*"activewindow"* [*texttosend*]]

Defines the alternate actions that an OKScript button performs when right clicked. Optional parameters are the same as the optional parameters of the BUTTON directive described above. Additional actions may be listed below the RIGHT directive as with BUTTON. Here is an example:

```
BUTTON hi red OKbridge Welcome everyone{enter}
RIGHT
  MESSAGE Welcomes players
```

NEWLINE

Causes subsequent button declarations to be placed on a new line in the button panel. In Plagiarize mode a NEWLINE directive is inserted by clicking on the **Newline!** menu item.

MENU *"menu path"* [*"window"* [*text*]]

MENU works exactly like BUTTON except it creates a menu entry instead of a button. The *"menu path"* has the form: *"text1|text2|...|textn". Text1* is the text that appears on the menu bar. The remaining *texts* are the cascading menu entries you want built for this script. Setting the last menu entry in *"menu path"* to a hyphen (-) creates a separator line in the menu, e.g.:

```
MENU abc|def
MENU abc|-
MENU abc|ghi
```

A keyboard activation key for any menu name can be specified by preceeding the character with an ampersand (&), e.g. "&Greetings|&Welcome". (To include an ampersand in the name, use &&.)

Menu paths should generally not branch off of a previously created menu path. Consider:

```
- Example of improper menu structure
MENU abc|def OKbridge bla bla
MENU abc|def|ghi OKbridge bla bla
```

An error will be indicated when OKScript gets to the second line because `abc|def|ghi` adds

to an existing full menu declaration - `abc|def` - so the first declaration, and any actions that go with it, are inaccessable and reflect flawed logic somewhere in the script's design. (Strictly speaking, it may at times be useful to include a menu stub such as `MENU abc|def` in a script as a way to logically position a menu branch at a particular point in a script. By including PRAGMA MENUSTUBS OKScript will accept such constructions.) Similarly it is also improper to declare a menu name that is a subset of an existing menu, e.g.:

```
- Example of improper menu structure
MENU abc|def
MENU abc
```

Note that it IS ok (and common) to have a menu name that uses a portion of an existing menu, as illustrated by:

```
- Typical menu branching
MENU abc|def OKbridge bla bla
MENU abc|ghi OKbridge bla bla
```

MENUs and BUTTONs can be mixed freely in a script.

Input and Output

message - window - put - toclip - get - decode - Notation and Layout

MESSAGE *text*

Message displays a dialog box containing *text.* MESSAGE passes all metacharacters literally except {enter}, thus making multiple line messages possible, for example:

WINDOW *windowname*

Activates the window whose title bar contains the string text or window handle specified. If WINDOW fails to find *windowname* OKScript behaves according to the ONERROR flag.

PUT *text*

Sends *text* to the active window. This text is subject to macro substitution and may contain OKScript metacharacters that allow any compound keypress to be coded and sent to an application, e.g. ctrl-E.

```
MESSAGE 1st of 2 lines{enter}display plus {+} literally.
```

A script can be terminated at this point by clicking on the dialog's **Abort Script** button. Message boxes grow to accomodate long message lines.

TOCLIP *text*

Puts the *text* argument in the clipboard. Often this will contain a macro reference, e.g.: `TOCLIP %xxx%`.

GET *macroname prompt*

Prompts the user for input that will become the definition of the macro with the name *macroname* if the dialog is completed normally, which also sets the -CANCEL macro to "0". Clicking the **Cancel** button will abandon this assignment and sets the -CANCEL macro to "1". To terminate the script at this point click on the dialog's **Abort Script** button. If the *prompt* field contains an {enter} then everything beyond that point is added to a pulldown list attached to the input control. In this way a script can offer choices to the user, for example:

```
GET result Select a number{enter}1{enter}1{enter}3
```

Note that the {enter}s must not be generated through a macro. This button exhibits the desired behavior:

```
MACRO e {enter}
BUTTON test red
  LET acc Select a number
  LOOP 3
    LET acc %acc%{enter}%-indexup%
    POOL
  MESSAGE %acc%
  GET x %acc%
```

However, change the line in the loop to:

```
- this line fails
LET acc %acc%%e%%-indexup%
```

and it fails because LET does not expand metacharacters contained in macros, specifically the %e% is not be expanded to the proper internal representation. LET can be forced to do this by prefixing the target macro with an asterisk (*), e.g.:

```
LET *acc %acc%%e%%-indexup%
```

DECODE *encodedtext*
DECODE performs the same function as PUT (see above) except the text that is sent is decoded first. To encode a text string (such as a password) use the Encode a Password utility in the Script menu. Unlike the PUT instruction, DECODE accepts the OKScript metacharacters literally. Decode uses the window activation string from the most recent window activating instruction (MENU, BUTTON, RIGHT or WINDOW) as part of the decoding process. It is important that this string match the string that was entered in the Encode form to produce the *encodedtext.* DECODE is useful for concealing sensitive text from casual observers of a script. It should not be considered secure as it is not difficult to uncover the underlying password.

Procedure (Subroutine) Features

proc - call - Notation and Layout

OKScript provides a minimal set of control flow instructions to facilitate advanced application scripting. Iteration (loops) and conditional (IF THEN ELSE) capabilities are also provided and discribed in separate help topics.

PROC *procname*

Defines the entry to a procedure. The lines following the PROC directive are executed whenever `CALL name` is encountered in a script. As with a BUTTON, a PROC code is completed when a terminating declaration line is encountered, at which point execution resumes with the line immediately following the CALL action. The case of a PROC's *name* is ignored. PROC calls can be nested to a depth of 10 and are only accessable to the current panel or a panel that has a USE instruction that references this panel. A PROC may access a parameter string from a CALL instruction (see below) through the predefined macros -args, -arg1, -arg2, through -arg9. These arguments arrive with macros expanded but no metacharacters translated. If you expect that there might be metacharacters in the argument string you will need to translate them with a LET * instruction.

Here is an example of a PROC that sends it's parameter string to the Notepad. It is called by a BUTTON that then sends other text.

```
PROC PutSomeText
  LET *x %-args%
  PUT %x%
BUTTON ProcTest red Notepad
  CALL PutSomeText ,text_from_proc
  PUT text_from_button
```

CALL *procname arguments*

Invokes the specified PROCedure. When the procedure completes its execution control is returned to the instruction following the CALL. The case of a PROC's *name* is ignored. Macro calls embedded in the argument field are expanded but metacharacters are passed untranslated. This allows metacharacters in arguments to be successfully passed to another procedure. To use an argument containing a metacharacter in a PROC assign the argument to a macro the LET * instruction. See the example in the PROC description above. PROC calls can be nested to a depth of 10.

The *arguments* string can be accessed from within the PROC through the predefined macros -args, -arg1, -arg2, through -arg9. The first character of *arguments* is the delimiter character that divides up the rest of the string. Consider this call to aproc:

```
  CALL aproc ,xxx,yyy,zzz
```

Within the PROC aproc, %-args% returns xxx,yyy,zzz, %-arg1% returns xxx, %-arg2% returns yyy and %-arg3% returns zzz.

OKScript Control Functions

open - close - topanel - mode - pragma - stop - Notation and Layout

These instructions control the operation of the OKScript program itself.

OPEN *[*]scriptfile*

Reads the specified script file into a new editor page unless the file is already loaded. The optional asterisk (*) preceeding the filename replaces the file if it is already loaded; if it is the currently running file it's panel is rebuilt. OPEN is typically used in a script that loads a collection of related scripts into different editor pages. If a directory (folder) is specified for *scriptfile* then all the .oks files in that directory will be loaded into different editor pages. It is recommended that the script's full pathname be specified - a good application for a macro. If you don't want this setup script to appear in the tabset it can be removed with a terminating CLOSE action. E.g.

```
MACRO dir c:\oks40\
STARTUP
  OPEN %dir%greetings.oks
  OPEN %dir%2over1.oks
  OPEN %dir%sayc.oks
  CLOSE
```

CLOSE *[next]*

Closes the current script page and moves to the script page named by the *next* argument. If there is no *next* argument, or that file isn't loaded, then OKScript advances to the next page. If there is only one file loaded, CLOSE will remove it and enter the script editor. Actions that follow a CLOSE action are ignored.

TOPANEL *tabname*

Causes the script stored in the editor page whose tab label is *tabname* to be compiled and replace the current button panel. This is useful for invoking panels that are logically subordinate to some function in the current panel, e.g. a bidding sequence. Actions that follow a successful TOPANEL action are ignored. A TOPANEL to a non-existant panel is an error that is processed according to the ONERROR state.

MODE TITLE *name*
MODE STATUS *color text*
MODE <ON | OFF | FLIP> <MENU | BUTTON> *"name"*
MODE <ON | OFF | FLIP> MENUCHECK *"name"*

The first form of the MODE instruction changes the filename part of OKScript's title bar. This value will persist until renamed or the file is reloaded.

The second form of the MODE instruction sends *text* to the status bar in the color specified. See BUTTON for a list of colors. See PRAGMA (below) to learn how to create a status bar.

The third form of the MODE instruction allows you to disable or re-enable a script-created button or menu. A disabled control appears "grayed out" and cannot be activated. This feature can be used as an alternative to the plagiarize feature to dynamically build custom script configurations from a "base script." The first field commands the control to be enabled (ON), disabled (OFF) or to reverse its current state (FLIP). The next field specifies what the control is, a BUTTON or MENU. The last field is the caption that appears on the control. In the case of menus this can be a partial name, for example if a script contains:

```
MENU foo|bar|zzz
```

then the instruction:

```
MODE OFF MENU foo|bar
```

is acceptable and will leave "foo" enabled but will disable "bar" and "zzz".

The final form of the MODE instruction controls the check mark beside a menu item. OFF removes the check mark, ON adds the check mark and FLIP reverses the check mark state.

Note that all forms of the MODE instruction allow the ON / OFF ... field and MENU / BUTTON ... field to be specified with a macro, thus:

```
MODE %a% MENU foo|bar
```

PRAGMA WINERROROFF
This pragma disables the reporting of window activation errors which can occur when executing WINDOW, BUTTON, MENU or RIGHT instruction. It applies to the whole panel, regardless where it is located in the script and should be used with care.

PRAGMA FILEHELPOFF
This pragma makes a panel's File and Help menus invisible and should appear near the top of a script. The hot-key functions of these menus are still available, even though invisible; in particular, hitting Ctrl-E will return to the script editor.

PRAGMA STATUSBAR
This pragma causes a status bar to appear at the top of a button/menu panel. You can display information in this status bar with the MODE STATUS instruction (see above.) This pragma must be placed in your script before any BUTTONs.

PRAGMA <MENUSTUBS | MENUPROPER>
This PRAGMA inhibits (MENUSTUBS) or enables (MENUPROPER) error reporting for two questionable menu declaration forms, specifically, menus that add to the full declaration of an existing menu, e.g.

```
MENU abc|def
MENU abc|def|ghi
```

and menus that do the reverse:

```
MENU abc|def|ghi
MENU abc|def
```

Thses forms are generally incorrect, but may be necessary in the construction of certain plagiarization and other libraries. See the MENU description for more information.

STOP
Causes the OKScript program to terminate.

OKScript Language Reference

Notation and Layout

For an overview of OKScript concepts and usage please see the introduction and tutorial. The reference pages list instructions by typical frequency of use.

Language Summary

Instructions by Function

General Features

How do I get the File Tabs to display?

Sometimes the file tabs at the bottom of the OKScript window do not appear properly, or at all. There are two things that can go wrong. First, check that tab display is enabled: in edit mode hit Script | Set Panel Options and under the Default tab make sure that 'show' is selected in the Filetabs item. Next, if your script has a PLACE instruction make sure that it's third argument is either 1 (show the tabs) or -1 (use the global panel options setting.)

The second thing that can go wrong is that your panel is so narrow, or file names so long, that they don't fit in the alloted space. If there is a short file name in your set of loaded scripts, often that file will be the only one in a visible tab. Try clicking on the little arrow icons in the lower right which cycle through the tabs. If that doesn't work all that is left to do is drag OKScript wider or rewrite your script so it displays wider. Note that you can move from one panel to another even if the tabs are not visible by hitting ctrl-tab and ctrl-shift-tab.

Can I get more than one row of buttons?

Yes, just separate the BUTTON directives that you want in separate rows with the NEWLINE directive. If you are building a panel by plagiarizing you can insert this by clicking on the **Newline!** menu.

Performing DOS Operations

Many DOS file and directory functions are built into OKScript. Other DOS functions can be performed by having the OKScript RUN action invoke the DOS command interpreter, COMMAND.COM.

First you need to locate COMMAND.COM on your system. Windows 9x puts this file in the c:\windows folder. Windows NT usually puts it in the c:\winnt\system32 folder. If you can't find it do one of the following:

- In a DOS window type the word SET. In the list there should be a line that looks like:

  COMSPEC=xxx

  where xxx is the path for COMMAND.COM

- Alternatively, use the Windows find function in the Start menu or use File | Search in the Windows File Manager to locate COMMAND.COM.

Once found, you can create a script that performs anything that you can do in DOS. Here is a sample script line (assuming for the moment Windows 9x) that copies a.oks to b.oks in the c:\oks40 directory.

```
RUN c:\oks40 c:\windows\command.com /c copy a.oks b.oks
```

Tutorial page 2

Plagiarizing Scripts

The simplest way to create a button panel is to use pre-defined buttons. Click the **File | Open** menu item and load 'Palette.oks'. Next hit **File | Plagiarize | To-New-Page** then the Palette.oks tab. Click a few buttons and click the tab of the new (unnamed) editor page. Voila, you have just created your first button panel. Hit **File | Edit Panel** to see the scripts that were produced.

Note, if you hit **Newline!** (in the menu bar) while plagiarizing any buttons that are now pressed will be placed in a new row of the button panel. **Newline!** inserts a NEWLINE directive in the script being produced. You can examine the script that was produced by clicking **File | Edit Panel**.

You can plagiarize custom menus in the same way as buttons. To learn about OKScript's advanced block plagiarize feature, see the PLAG/GALP discussion in the reference section.

Next - writing one line scripts

Tutorial page 3

Writing One Line Scripts

Next you will learn how to actually write some basic scripts. First, if there is a button panel on the screen, switch to Edit mode by clicking **File | Edit Panel**. Next close all the open editor pages with **Edit | Close All**, then open a new blank page with **File | New**. In the editor type:

```
BUTTON FirstButton red
```

Now hit **ctrl-R** (or **Run Panel** in the **File** menu) and a button bar will appear with one button. This button does nothing, but it's a start! The first two parameters of the BUTTON directive are pretty obvious, a name to display on the button and it's color. Hit **ctrl-E** (**File | Edit Panel**) to return to the editor and modify this line so that it reads:

```
BUTTON "First Button" red Notepad It Works!
```

Note that it is fine to have button names with spaces, but such names need to be "quoted". The third parameter is the name of a window that you want activated when the button is pushed. It is case sensitive, but only needs to match a portion of the window's title. The rest of the line is text that is sent to the activated window. Hit **ctrl-R** and your button panel will reappear. Now hit the button you just built. The message "It Works!" should appear in Notepad's text area.

Note that these examples use Notepad as the target. Scripts designed for OKwin need to replace "Notepad" with "OKbridge" as the target.

In addition to buttons, scripting instructions can also be attached to cascading menus. Try running your script when it is attached to a menu:

```
MENU "User|First Button" Notepad It Works!
```

Typically a very high portion of custom buttons and menus are of this basic, one line form. Let's play with our button a bit more. Modify your script (those are **braces**, not parentheses) so it reads:

```
BUTTON "First Button" red Notepad It{enter}Works!
```

and try it out. Notice how you can send multiple lines by inserting {enter}. All the non-printing keyboard keys can be sent by enclosing their name in braces. See the metacharacters discussion for details on sending special characters, including those specific to OKbridge.

Next - multi-line scripts and panels with multiple rows of buttons

Tutorial page 4

Multi-line Scripts

Occasionally it will take more than one line of code to do all that your button or menu to do. Try this:

```
BUTTON Second blue Notepad
 PUT one line{enter}
 PUT another line{enter}
```

OKScript knows to end a script when it hits a non-executable directive (other than comments, which are ignored.) These are usually a NEWLINE, another BUTTON (or MENU) directive, or RIGHT. Here is an example.

```
BUTTON Second blue Notepad
 - comments don't terminate a button's definition.
 PUT one line{enter}
 PUT another line{enter}
 - the following button DOES end it.
BUTTON Third green Notepad
 PUT funky line{enter}
 PUT plain line{enter}
 - this ends the button "Third"'s declaration.
NEWLINE
BUTTON NewLineButton black
```

This script produces three buttons on two lines. The NEWLINE directive starts a new line of buttons in the button panel. Try it.

When a script contains instructions that are very long and ungainly they can be divided into multiple lines with the MORE instruction. For example:

```
MESSAGE bla bla bla bla bla bla
```

can be rewritten:

```
MESSAGE bla bla
  MORE bla bla
  MORE bla bla
```

Each MORE instruction inserts exactly one space between the preceeding line and itself.

Next - using macros

Tutorial page 5

Using Macros

The last thing this tutorial will cover is macro substitution. Macros allow scripts to be dynamically customized. Here is an example. Hit **ctrl-N** (**New** in the **File** menu) and enter:

```
 MACRO pd Cathy
BUTTON glp red Notepad Good luck %pd%
```

Next build this panel (**File | Run Panel**) and hit the button. 'Good luck Cathy' will appear. The %pd% is replaced by the current definition of the macro 'pd' at the time the button is pushed. You can change a macro's definition dynamically (after the panel is built) with the GET and LET actions. Here is an example of how you might put this to use:

```
 MACRO pd Cathy
BUTTON "Enter pds name" black
GET pd What is partner's name?
BUTTON glp red Notepad Good luck %pd%{enter}
```

You would use the first button once, at the start of a partnership, to input partner's name to OKScript, and the second button to issue a good luck message customized for him/her. Remember, to use these scriptlets with OKWin replace "Notepad" with "OKbridge".

Next - making a shortcut (icon) for your scripts

Tutorial page 6

Creating an Icon for Your Scripts

When you finish a script that you want to reuse be sure to save it by clicking **SaveAs** in the **File** menu and picking a meaningful .oks name for your script file.

Once you get a panel configuration that you like you can create a Windows shortcut (icon) to load that configuration from the desktop or Start Menu. (This discussion assumes the 32 bit version of OKScript. Go here to learn how to do this in the 16 bit Windows 3.x world.)

There are two ways to make an icon that loads your files. The first is to modify the shortcut that you use to launch OKScript now. To do this right-click the icon and select 'properties' from the popup menu, then select the shortcut tab. On the "target" line you will see something like:

```
"c:\oks40\okscript.exe" /e examples.oks
```

The "/e" causes OKScript to start in the editor, remove the "/e" and a button panel will be built immediately on launch. Change the last part of this line so it contains the files that you want loaded, something like:

```
"c:\oks40\okscript.exe" file1.oks file2.oks
```

You can list as many script files as you want. The last file will be the panel to appear when you launch OKScript with this icon.

Here is the easy way to create a new icon for a particular panel configuration. First, load all the files that you want and run (build) the panel, then click **File | Make Shortcut** and fill out the dialog. Clicking OK will create a custom icon on your desktop or in your Start Menu.

Next - reducing the OKScript footprint

Inserting and Splitting Script Files

The OKScript editor does not directly support these functions, however they can be easily achieved by using another editor page to temporarily hold a file.

Inserting (merging) one file into another

First read the file to be inserted into a new page by clicking File|Open an selecting the desired file. Next copy this file to the clipboard by selecting all of it (Edit|Select All) and hitting ctrl-C. Finally position the cursor at the location in the target file where you want the insertion to take place and hit ctrl-V.

Splitting (extracting from) a script file

Select the section of the file that you want to extract out, either by dragging the mouse over it or moving the cursor over it with the arrow keys while holding the shift key down. Next hit ctrl-X to put it in the clipboard. Finally open a new editor page (File|New) and insert the clipboard with ctrl-V. Save this page to the file of your choice.

Scheduling Directives

Notation and Layout

The DO directives cause scripts to be executed when some time related or user event happens.

DO AT *date time*
DO EVERY *time*
DO IN *time*

Causes the actions following the DO line to be executed when the specified time event occurs. If another script is executing at that time the DO actions are deferred until it completes. The date field is optional, and the current date is assumed if absent. The format of the date and time fields are specified according to your computer's regional settings. Each script file can have up to 10 DO directives.

DO AT causes the actions to be executed at the specified time

DO EVERY causes the actions to be executed repeatedly with a period specified by *time*.

DO IN causes the actions to be executed after the specified *time* has elapsed.

DO <BEFORE | AFTER> <MENU | LEFT | RIGHT> *eventname*

Causes the actions following the DO line to be executed whenever the user clicks on a script defined MENU or BUTTON. In the BEFORE case the following instructions are exectued before the control's normal functions. The AFTER case's instructions are executed after the control's normal code is completed. LEFT and RIGHT allows you to distinguish between left and right button clicks. Attributes of the activating control are accessable via the -MYNAME, -MYCHECK and -MYCOLOR predefined macros.

One use for this feature is to display the last relevant button pressed in a status bar. (The system already underlines the last button that was pressed, but there is no way to have 'utility' buttons that are excluded.) In the sample below, relevant buttons are red and the rest are ignored.

```
- this line causes your panel
- to display a status bar
PRAGMA statusbar

- this code displays the name of the
- last *red* button that was pressed in
- the status bar. non-red buttons have
- no effect.
DO AFTER LEFT
  IF NOCASE %-mycolor% EQ red
    MODE STATUS red last button: %-myname%
    FI

- here are some empty buttons to test   - the above code
BUTTON test1 green
BUTTON test2 red
BUTTON test3 red
```

How was OKScript implemented?

For the curious and technically inclined, this implementation of OKScript consists of about 5500 lines of Borland (now a part of Inprise Corp.) Delphi, an object oriented Pascal dialect, code.

Uninstalling OKScript

Some distribution packages for this release of OKScript include support for automatic uninstall. To determine if your installation does, open the Windows Control Panel - Add/Remove Programs applet from the Start Menu. If OKScript is in the list of installed programs simply select it and click the Add/Remove button. (Note that this procedure removes only the files that were a part of the original setup. It does not remove any scripts or shortcuts (icons) that you produced, or configuration and registration information.) If OKScript isn't in the list, or if you want to remove all the files that the automatic uninstall leaves behind, read on ...

The basic OKScript installation process does not modify the Windows Registry, .INI files or install any files outside of the OKScript installation directory except any shortcuts (icons) that may have been produced (on 32 bit systems only.) To uninstall OKScript simply locate that directory (using the Windows Explorer or File Manager) - c:\oks40 by default - and delete it.

If you have trouble locating the directory, you can examine the shortcut (icon) that you use to launch OKScript (in Windows 9x and NT) by right clicking on it and selecting 'properties' from the popup menu. In the resultant dialog click on the 'Shortcut' tab. The 'Target' field contains the name of OKScript directory.

Script Menu Reference

encode - remove blanks - nesting check - beautify - options

The Script Menu provides access to functions that add to, check and format your script code. The Script Menu is not available in run mode.

Encode a Password

Passwords can be protected in a script by encoding them. Click on this item and enter the password and the text you will be using to activate the target window for the password in your script then click 'OK'. The encoded form of the password will be placed in the clipboard where it can be pasted into the script. The text to be encoded may contain any printable ASCII character except space. Encoded passwords are decoded by the DECODE instruction. This feature useful for concealing sensitive text from casual observers of a script. The inclusion of the target window string makes it more difficult for unauthorized individuals to discover the underlying password. Even so, such encoding should not be considered completely secure as it is not impossible to uncover the underlying password.

Remove Trailing Blanks

Removes blank (space) characters at the end of script lines in the current edit buffer. (The script engine ignore trailing blanks so this is just a cosmetic, space saving function.)

Check Nesting

Checks for the proper nesting of the multi-line constructs, including IF ELSE FI, LOOP POOL, PROC, STARTUP AND BUTTON, in the current script file. This check is also performed at the start of Beautification, see below.

Beautify Code

Attempts to improve the readability of the current file by capitalizing all the OKScript instruction names and indenting the code in a consistent, uniform manner that reflects the script's structure. Beautify also performs a nesting check, see above. Note, you may want to save, or copy the file to the clipboard, before beautifying it, just in case you don't like the outcome. (In case you are wondering why your disk is accessed, the beautify operation writes and deletes a temporary file to the OKScript directory.)

Set Panel Options

This displays a notebooked dialog box containing options for hiding or showing the file tabs on button panels, enabling 'window rollup', and setting the keyboard button navigation mode and editor size. The first two of these features let you balance panel appearance with its screen footprint. Keyboard navigation allows you to select panel buttons without a mouse, using the keyboard. These features are discribed below.

The controls in the first notebook page establish the default settings for file tab display, rollup and navigation. The second page sets the tab display and rollup for the current script (over-riding the defaults) by adding (or replacing) a PLACE instruction that reflects the settings to be used only for that script. The PLACE instruction will also contain the screen position of the last button panel that was displayed.

If tabs are displayed the user can switch between panels by clicking the tab of the desired script. If tabs are disabled the user can still switch between pages, but must use **ctrl-tab** and **shift-ctrl-tab** or **File | Page Fwd** and **File | Page Back**.

When rollup is enabled OKScript "rolls up" into just a window title bar when another application

is active and "unrolls" when it becomes active, e.g. by clicking on it. There is a small pause before rollup when OKScript becomes inactive as a result of a script execution.

Normally scripts are executed when you click on a button. Keyboard navigation enables button selection via the keyboard. With matrix addressing a button's row and column numbers are specified by entering two digits. With spreadsheet addressing a letter and a digit specify a button's column and row position. In both cases the top row and leftmost column is numbered zero, not one. Using matrix addressing, activating the last button in a panel with four rows and three buttons in the last row would be effected by hitting 32 and <enter>. Using spreadsheet addressing hit c3 and <enter>.

Global-only Settings

Sometimes a keystroke receiving application cannot keep up with the character stream produced by OKScript, producing garbled results. The "Pause after sending ..." option lets you insert delays after every character sent or only after {ENTER}s by checking the appropriate box. Click here to learn more. These delay settings will persist between sessions.

For really big scripts, the 32 bit editor can be configured to accomodate 128k, 256k or 512k. This setting affects the current editor and any subsequently opened editor.

Help Menu Reference

The Help menu provides access to this help file as well as program information and license code entry.

Tutorial and Reference (F1)
This opens the OKScript help file, which you are currently reading.

Register
Use this function to enter the registration name and code provided when you registered OKScript. Be sure to record and keep this information in a safe place, you may need to re-enter it if you upgrade the software. (This won't be necessary if the upgrade is placed in the same directory. The license information is stored in the OKSCRIPT.INI file.)

About
Displays the About box which contains version and registration information about your copy of OKScript.

File and Directory Actions

onerror - file - path - name - expand - find - dir - Notation and Layout

All file and directory manipulation functions are performed with variations of the FILE or DIR actions described below. Errors are handled according to the setting of ONERROR, see below.

ONERROR *flagvalue*

OKScript uses an internal flag to determine what action it should take if an error occurs while running a script. The user can set this flag with the ONERROR action to either REPORT, FLAG or FLAGONCE. If not specified, the default value is REPORT. In general, if an operation that can produce an error is successful the macro -err (accessed with %-err%) is set to zero ('0') otherwise it is set to an error code, usually '1'. The IF instruction can test -err. If the operation produced an error, and the ONERROR flag is set to REPORT, the script is terminated and an error message is displayed. Regardless of the success of the operation if the ONERROR flag is set to FLAGONCE it is reset to REPORT, otherwise the ONERROR flag is unchanged. The ONERROR flag is also used by the RUN and WINDOW actions.

FILE DELETE *filename*
FILE RENAME *"oldfilename" newfilename*
FILE MOVE *"sourcefilename" destination*
FILE COPY *"sourcefilename" destination*

These instructions perform the specified operation (delete, rename move or copy) on the indicated file(s). MOVE and COPY replace the destination file if it already exists.

FILE PATH *"macroname" filepath*

Copies the directory part of *filepath* to the macro specified. The resulting string will end with a backslash (\) character if there is a backslash present anywhere in *filepath*. FILE PATH cannot produce an error.

FILE NAME *"macroname" filepath*

Copies the filename part of *filepath* to the macro specified. FILE NAME cannot produce an error.

FILE EXPAND *"macroname" filepath*

Expands *filepath* into a fully qualified file specification and place the result in the macro specified. FILE EXPAND cannot produce an error.

FILE FINDFIRST *"pathmask" attributes*
FILE FINDNEXT
FILE FINDCLOSE

FINDFIRST begins a search for files in the directory and with names matching *pathmask* that have the characteristics selected in *attributes. Pathmask* is a standard file selection mask of the form c:.oks. *Attributes* is a number created by adding the desired values from the following table:

Value	File type selected...
1	Read-only files
2	Hidden files
4	System files
8	Volume ID files
16	Directories (folders)
32	Archive files

63 Any file

The result of the search is accessed in the predefined macro -FOUND. This will be empty if no files were found. Additional files matching this specification may be sought with the FILE FINDNEXT instruction. A FILE FINDCLOSE instruction must be issued when searching has been completed. FINDFIRST instructions may be nested up to 10 levels deep to perform sub directory searches. Here is a script to display all the .oks files in the OKScript directory:

```
FILE FINDFIRST c:.oks 63
LOOP 10000
  IF TEXT %-FOUND% EQ
    BREAK
    FI
  MESSAGE %-FOUND%
  FILE FINDNEXT
  POOL
FILE FINDCLOSE
```

DIR CHANGE *path*
Changes the current directory to *path.*

DIR MAKE *path*
Creates a directory named *path.*

DIR REMOVE *path*
Removes the directory named *path.*

DIR GET *drivecode macroname*
Puts the current directory path for the drive specified into the macro named *macroname. Drivecodes* are either the drive letter, optionally followed by a colon, e.g. **c** or **c:**, or are numbers that associate with drive letters in the sequence: 1 = a:, 2 = b:, etc. Drive number 0 is the current drive.

Tricks for Windows 3.x Users

To simplify access to OKScript from Windows 3.x you may want to create a link to it in the Windows 3.x Program Manager and associate the .oks script file type with OKScript so that double clicking on an OKS file will launch OKScript. Here are the procedures:

To add an OKScript icon to some group in Program Manager do this:

a. Select the group you want the icon to appear in, then
b. Click the **File | New** menu.
c. Select "Program item" in the dialog box that appears.
d. Click "ok".
e. Fill out the information requested:
   - Description: OKScript
   - Command Line: c:.exe
   - Add any script files you want loaded here too.
   - Working Directory: where your script files will be stored.
f. Click "ok".

To associate ".oks" files with OKScript so that you can double-click on one of these files and it will automatically startup OKScript, load the File Manager and:

a. Locate the OKScript directory in the File Manager.
b. Select (single click) a script (.oks) file.
c. Click the **File | Associate** menu item.
d. The dialog should have OKS in the "files with extension" field.
e. Hit the "Browse" button and select OKScript.exe and hit "ok".
f. Hit "ok" in the Associate dialog box.

Meta Characters

button/menu names - keystrokes - okbridge

In Button and Menu Names

An ampersand (&) in the name field of MENU and BUTTON directives causes the letter following the ampersand to be shown underlined. This character acts as a hotkey for this item - you can activate it by hitting alt-x, where x is the underlined letter. Use && in these name fields to override this feature and actually display an ampersand on the button or menu.

In Keystroke and Other Text

Most Windows applications provide functions that are activated by key sequences that include one of the "shift" keys - Alt, Ctrl and Shift. For example, an application's menu functions are accessed with "alt" key sequences. These shifted keys cannot be directly typed into the script editor, yet is is reasonable to want to send an alt-F to access the File menu. In OKScript the three shift keys are encoded by prefixing a character with a metacharacter - the tilde (~) for Alt, carat (^) for Ctrl, and the plus sign (+) for Shift. Here are three examples illustrating how these metacharacters can be used to send alt-F, ctrl-F and shift-ctrl-F:

```
PUT ~f
PUT ^f
PUT +^f
```

This shifting function is performed ONLY by instructions that send an argument to an application as keystrokes - the BUTTON, MENU, RIGHT and PUT instructions. (DECODE does not meta translate its argument.)

OKScript uses braces, { and } (note braces are NOT parentheses), to send other keys that are otherwise difficult to represent, including the shifting metacharacters themselves. For example, to send the string "16+ points, any shape" to the active window you would write:

```
PUT 16{+} points, any shape
```

Use braces to 'literalize' the macro delimiter character (% by default) and insert special key codes, such as {enter} (see below.) Braces surrounding a single character send that character literally. Use this to send a leading space (which OKScript would otherwise strip out.) Note that this translation only occurs in instruction arguments where macros are allowed.

Should the need arise, strings containing these devices can be translated into the intended characters with the LET * command. This is particularly useful when processing data destined for a MESSAGE. (Note that metacharacters that are in a macro definition are retranslated by the keystroke instructions, and therefore perform their meta function automatically, after macro is expanded.)

Here are the remaining keyboard keys that are sent by enclosing their name in braces:

Script code	Produces this key...
{BKSP}	Backspace
{TAB}	Tab
{ENTER}	Enter (sometimes called the Newline key)
{ESC}	Esc or Escape
{UP}, {DOWN}, {LEFT},	Cursor keys

{RIGHT}	
{HOME}, {END}, {PGUP}, {PGDN}	More cursor keys
{INS}	Ins or Insert
{DEL}	Del or Delete
{F1}, {F2}, ... , {F12}	Function or 'F' keys
{QC}	the current macro quote character, which is percent (%) by default. See the QUOTECHAR instruction for details.)

OKbridge Special Characters

In OKbridge, messages are sent to specific players by prefixing them with a message directing character. For example, to send an alert to both opponents start the message with the 'equals' sign (=):

```
BUTTON 1c red OKbridge =could be just 2 cards{enter}
```

Here are all the special OKbridge prefix characters:

	right hand opponent
	left hand opponent
	both opponents
	the spectators (needs to be in {braces} to not interfere with macros)
	the lobby
	everyone at the other table of a team game
	a specific player, e.g.: -joe{tab}hi joe{enter}

Transient Menus

There are two top level menu entries that are normally not visible. These menus appear only in specific circumstances which are documented below.

Newline! (Alt-W)

Newline! appears only when OKScript is in plagiarize mode. It causes a NEWLINE directive to be inserted into the script that is being created.

Abort_Script!

This menu item appears only after a script has been running for a few seconds, suggesting that the script may be in an infinite loop or is having some other problem. Clicking on **Abort_Script!** will force the termination of stalled scripts most of the time.

Self-Positioning Panels

The PLACE directive causes a script to produce a panel at a specific location on the screen. PLACE takes four arguments, the horizontal and vertical screen coordinate (in pixels) of the upper left corner of the panel, a flag that enables or disables the display of the panel selection tabs and a flag that enables or disables panel rollup. For example:

```
PLACE 100 100 1 -1
```

To automatically insert this directive the current script file use the **Scipt | Set Panel Options | In This Script** function. The resulting PLACE will replace any PLACE directive already in the file and will reflect the last location that a panel was positioned.

These instructions define the end of a preceeding declaration block:
BUTTON, MENU, RIGHT, STARTUP, NEWLINE, PROC, DO, PLACE, PRAGMA, USE, GALP, MACRO and QUOTECHAR.

Notation

Italics - the reference name of an instruction argument that is to be replaced by some meaningful value within an actual script, for example, the description: **WINDOW *windowname*** might become **WINDOW Notepad** in your script.

"Quoting" - an argument that accepts enclosing "quotes". Argument quoting is only required when the argument contains space characters. Use two quotes to embed a quote character in a quoted argument, e.g. "Contains "" a quote".

Underlining - an argument that ignores macro references. All other arguments (but not instruction names) expand macro calls.

[Brackets] - an optional argument.

<Angles> - a required choice, e.g. <ON | OFF>.

Layout

- Case-sensitivity - only window specifiers, case is ignored elsewhere.
- Indentation - ignored.
- Annotation - lines beginning with a hyphen (-) are ignored.

Any long script line can be divided into more managable units with the MORE instruction, e.g.:

```
MESSAGE bla bla
  MORE bla bla
  MORE bla bla.
```

Each MORE instruction adds exactly one space and its text field to the preceeding instruction. The example above is equivalent to:

```
MESSAGE bla bla bla bla bla bla.
```

Integer Formulas

OKScript interprets instruction arguments that contain an empty macro call (i.e. %%, assuming the default macro delimiter) as arithmetic formulas. When executed they produce a text string representing the value value of the formula. For example, the script:

```
MACRO Weeks 3
BUTTON Days red
  LET x %%%Weeks%*7
  MESSAGE there are %x% days in %Weeks% weeks.
```

diplays "there are 21 days in 3 weeks." in a message box. The following operators manipulate 32 bit integers in normal left to right fashion with standard precedence:

Code	Precedence	Function
+	lowest	addition
-	highest	negation
-	lowest	subtraction
*	middle	multiplication
/	middle	division
MOD	middle	modulo division
AND	middle	bitwise and-ing
OR	lowest	bitwise or-ing
XOR	lowest	bitwise exclusive or-ing
NOT	highest	bit reversal
SHL	middle	left shift (by the right hand argument)
SHR	middle	right shift (by the right hand argument)
ABS(x)	highest	absolute value
SGN(x)	highest	the sign of the number (-1,0 or 1)
RAN(x)	highest	a random number in a range from 0 to x-1
(...)	highest	nesting

Window Names / Title Bars

OKScript works by keeping track of window names - the text that appears at the very top of each window in a colored bar. That area is called the window's "title bar".

The BUTTON, MENU, RIGHT and WINDOW instructions activate the window whose title bar contains text that matches the instruction's window field, so:

```
BUTTON glp red OKbridge Good Luck, pd
```

activates a window containing the text **OKbridge** (making it ready to receive input) and then sends the text "Good Luck, pd" to that window. If you change OKbridge to Notepad it will send

the text to the Windows if that application is running, thus:

```
BUTTON glp red Notepad Good Luck, pd
```

This technique is, incidentally, a good way to debug scripts offline.

Macro Features

macro - quotechar - let - trunc - - parse - state - Notation and Layout

MACRO *macroname macrotext*

Defines a text substitution macro. *Macrotext* is assigned to the macro named *macroname.* Subsequently, when a script containing the string *%macroname%* is executed that string will be replaced with *macrotext.* Note that metacharacters used in macro definitions are active when substituted as part of an output string, but treated literally otherwise.

A MACRO directive will NOT redefine an existing macro with the same name. *Macroname* must not begin with a hyphen (-) character.

Macro definitions are cleared when the script editor is activated, but persist when you switch between button panels with ctrl-Tab or shift-ctrl-Tab or click on a page tab. One set of macros can thus be used across many script pages.

The GET and LET (see below) actions allow the user to write scripts that alter the value of macrotext at run time. Some other actions store a success/failure flag in the macro named **-err**.

Macro substitution applies to all OKScript instruction fields EXCEPT:

- Instruction names (e.g. BUTTON, PUT.)
- The name field of the BUTTON directive.
- All the arguments of the PLACE, MACRO and QUOTECHAR directives.
- The macroname in the TRUNC and LET instructions.
- The function name in the DIR and FILE instructions.
- The argument to ONERROR.
- The MODE type and value fields.
- The PRAGMA argument.
- The predicate and condition type fields of the IF instruction.

Twenty predefined macros are provided:

Macro	**Inserts into the script...**
-CLIPBOARD	the current clipboard contents.
-CURDIR	the current directory path.
-DATE	the current date.
-TIME	the current time.
-TIC	OKScript's 0.5 second counter, starts at 0 on program startup.
-TICWIN	the handle (unique ID) of the window that was active on the last counter tic.
-ACTWIN	the handle (unique ID) of the window that OKScript last activated with WINDOW, BUTTON, MENU or RIGHT.
-TITLES	the window titles currently catalogued, separated by {ENTER}s.
-ERR	a '0' if the last I/O or TOPANEL operation was successful, a non-zero error code otherwise.

-CANCEL	a '1' if the last GET operation was canceled, '0' otherwise.
-RES	the residual string from the last TRUNC operation.
-EXE	OKScript's full path name.
-PANEL	the full path of this panel's script file.
-INDEXUP	the current LOOP index, starting at 1.
-INDEXDN	the current LOOP index, starting at the iteration count.
-MYNAME	the name of MENU or BUTTON that is currently executing.
-MYCOLOR	the color of the currently executing BUTTON.
-MYCHECK	the check status of the currently executing MENU.
-ARGS & -ARGn	(n=1,2,...,9) inserts a PROC argument.
-FOUND	the file name returned by the last FINDNEXT or FINDFIRST instruction.

These predefines are inserted into a script like any other macro, i.e. surrounded by the current macro delimiter, '%' by default. -TIME and -DATE are useful for scripts that create logs. Note, these predefines begin with the hyphen (-) character, thus a typical usage might be:

```
PUT today is %-DATE%
```

The empty macro call, %%, indicates that the current instruction argument is to be evaluated as an arithmetic expression, for example:

```
MESSAGE %%7*%weeknumber%
```

QUOTECHAR *character*
Changes the macro quote character from '%' to some other symbol. The new quote character cannot be a letter, number or OKScript metacharacter (tilde, caret, plus and left and right braces.)

LET *[<* | *#>]macroname text*
Replaces the contents of *macroname* with *text.* Metacharacters found in macros that are expanded in *text* will be passed on as-is unless the asterisk (*) is present, in which case they will be converted to the codes that they represent. If the macroname is prefixed with the '#' character then any macro references in the result will be expanded into their respective values. This expansion process will continue until there are no more macro references.

TRUNC *macroname count*
Retains the first *count* characters in *macroname* and moves the rest to the -res predefined macro.

PARSE *"delimeters" destmacro sourcemacro*
PARSE removes the left-most token from the contents of *sourcemacro* and places it in the

destmacro. Tokens are delimited by any character in the set *delimeters.* The case of alphabetic characters is ignored, except that an upper case 'Q' specifies the double quote character and an uppercase 'S' specifies the space character. Typical usage might be extracting words from a sentence, thus:

```
LET src The quick brown fox
LOOP 4
  PARSE S dest src
  MESSAGE dest
  POOL
```

STATE SAVE *statename*
STATE RESTORE *statename*
STATE ERASE *statename*
The STATE instructions manage a persistant store for all of a scripts macro values. Many stores can be maintained and are distinguished by their *statename.* SAVE saves the current state, RESTORE restores a previously saved state, not erasing any current macros that weren't saved at that time, and ERASE removes the values from a particular *statename.*

Condition Testing (IF-ELSE) Instruction

Notation and Layout

OKScript provides a minimal set of control flow instructions to facilitate advanced application scripting. Procedure (subroutine) and iteration (looping) capabilities are also provided and discribed in separate help topics.

IF ... ELSE ... FI

The linear flow of script execution can be altered with the IF-ELSE-FI construct, whose full form is:

```
IF iftype arg predicate [text]
    actions
 [ELSE [iftype arg predicate [text]]
    actions]
 ...
 FI
```

Control flow decisions are based on a predicate applied to one of the nine iftypes that test:

- the content (TEXT or NOCASE),
- numeric value (VALUE),
- or length (LENGTH) of *arg* versus the *text* field; or
- the user's response to a question (QUERY), or
- a file exists (EXISTS, e.g.: IF EXISTS okscript.exe EQ), or
- if an application is running (RUNNING), or
- if a particular MENU or BUTTON is enabled or checked (MENUCHECK).

The predicate specifies how the particular iftype test is to be performed. The QUERY, RUNNING, EXISTS, MENU and BUTTON iftypes only operate with the EQ and NE predicates. The IN and NIN predicates only operate with TEXT and NOCASE iftypes.

Predicate	**Returns true when...**
EQ	the test result is true or tested items are equal
NE	the test result is false or tested items are unequal
GT	*arg* is greater than the *text* field
GE	*arg* is greater than the *text* field
LE	*arg* is less than or equal to the *text* field
LT	*arg* is less than the *text* field
IN	the *arg* string is contained in the *text* field
NIN	the *arg* string isn't contained in the *text* field
HAS	the *text* field is contained in the *arg* string
NHA	the *text* field isn't contained in

S the *arg* string

The IF instruction performs the test and, if the result is true, executes the actions in the following lines. If an ELSE is encountered prior to a FI, the instructions from the ELSE to the FI that is associated with this IF are skipped. If the test evaluates to false then control moves to the ELSE part if present. An IF instruction may have more than one ELSE part. ELSE parts may specify further testing of the same sort. IF instructions must be terminated with a FI.

IFTYPES
The various iftypes use the IF instruction's text field in various ways to form a true or false result upon which to base a control flow decision. The *arg* and *text* fields can contain references to OKScript macros.

The TEXT and NOCASE iftypes compare *arg* with the IF instruction's *text* field. The NOCASE variation ignores the case of the comparison string. For example:

```
IF NOCASE %name% EQ Cathy
    MESSAGE this is Cathy
  ELSE
    MESSAGE this is someone else
  FI
```

The LENGTH iftype compares the length of *arg* with the *text* field.

The VALUE iftype converts the two fields into numeric values and compares them.

The QUERY iftype displays the *arg* field (a question, usually quoted) in a dialog box and lets the user select either YES or NO (or Abort Script.) A YES selection results in a true outcome with the EQ predicate. The *text* field is ignored. Here is an example:

```
IF QUERY "Continue this script?" NE
  MESSAGE Quitting
  FINISH
  FI
```

The RUNNING iftype produces a true result when a window can be found in OKScript's window database whose caption contains the *arg* field string. Note that windows that have not been active (focused) while OKScript has been active may not be successfully found by this test. You may need to click on the desired window before this test to ensure it is registered with OKScript. The *text* field is ignored.

The EXISTS iftype produces a true result when the file specified in the *arg* field is present and the predicate is EQ, or is not present and when the predicate is NE. The *text* field is ignored.

The BUTTON and MENU iftypes produce a true result when the particular control specified in the *arg* field is enabled and the predicate is EQ, or disabled when the predicate is NE. The *text* field is ignored. Similarly MENUCHECK produces a true result when the specified menu is checked and the predicate is EQ, or disabled when the predicate is NE.

Here are two examples of IF instructions:

```
IF NOCASE %-clipboard% EQ abcde
    PUT string is abcde
  ELSE NOCASE %-clipboard% EQ fghij
    PUT string is fghij
  ELSE
    PUT string is neither
  FI
```

Here is a button declaration that exercizes both LOOP and IF. It launches an application (Notepad) and waits for it to load before sending it a string.

```
BUTTON RunPad green
  - Launch Notepad
  - Note, the '.' in the RUN instruction below
  - indicates the current directory.
  RUN . Notepad.exe
  - Wait in a loop until Notepad is seen running
  LOOP 10000
    IF running Notepad eq
      BREAK
      FI
    POOL
  - Activate the just launched application
  WINDOW Notepad
  - Send it a some text
  PUT i'm ready now!
```

Note that the loop in this example could be replaced with a single instruction: WAITFOR Notepad. Also note that statements that follow a test for running have only partial assurance that the application in question will *still* be available - it is possible (though unlikely) that the application was terminated in the brief interval between the test and the subsequent instructions.

Miscellaneous Instructions

more - finish - waitfor - delay - run - plag - Notation and Layout

These are the OKScript instructions that don't fit in other categories.

MORE *text*

Any long script line can be divided into more managable units with the MORE instruction, e.g.:

```
MESSAGE bla bla
  MORE bla bla
  MORE bla bla.
```

All MORE lines are appended, with exactly one space character, to the preceeding instruction before is processed. Comment and blank lines are not allowed in a group of MOREs. The example above is equivalent to:

```
MESSAGE bla bla bla bla bla bla.
```

FINISH *[message]*

The FINISH action terminates the currently executing script. The optional message is displayed in a message box. FINISH is generally used in conjunction with IF to cause a script to stop when a particular condition is encountered.

WAITFOR *windowname*

This instruction causes OKScript to pause until a window whose title bar contains the text *windowname.*

DELAY *x*

Delays execution *x* milliseconds.

RUN *"dir" "prog-name" arguments*

Attempts to run the program *prog-name* with the initial directory *dir* and the parameter list *arguments*. *Dir* and *prog-name* must be "quoted" if they contain spaces. If RUN fails OKScript behaves according to the ONERROR flag.

Before sending keystrokes to an application that was launched with a RUN instruction it is a good idea to test to make sure that the application is, in fact, ready. This can be done by following the RUN instruction with a DELAY of sufficient duration. A better way is with a loop that tests for the application's presence, for example:

```
- Note, the '.' in the RUN instruction below
- indicates the current directory.
RUN . Notepad.exe
LOOP 9999
  IF running Notepad eq
    BREAK
    FI
  POOL
WINDOW Notepad
PUT whatever
```

In this example the "IF running" logic could also be replaced with the WAITFOR instruction.

PLAG
GALP
PLAG and GALP bound an extended collection of instructions that are to be captured on plagiarize. This might be used, for example, to capture an entire menu tree to a new panel, or a collection of support PROCedures. The PLAG instruction is inserted within a MENU or BUTTON declaration. A corresponding GALP is inserted after the last declaration to be plagiarized. Here is an example of how these might be used to plagiarize three MENUs (including Merge) when Conv|Jacoby|Merge is clicked:

```
MENU Conv|Jacoby|Merge
  PLAG
MENU Conv|Jacoby|2H ...
MENU Conv|Jacoby|2S ...
GALP
```

To plagiarize a collection of support routines use something like this:

```
MENU MergeSupport
  PLAG
PROC ...
PROC ...
...
GALP
```

PLAG and GALP blocks may be nested. They are ignored when not plagiarizing, except that GALP is a declaration block terminator.

Color names

BLACK - BLUE - SKY - GREEN - PURPLE - AQUA - MAROON - OLIVE - NAVY - TEAL - LIME - RED - ORANGE - PINK - YELLOW - GRAY - WHITE

Tutorial page 7

Reducing a Panel's Footprint

It is generally desirable to keep OKScript panels as small as possible. Here are some tips to achieve that.

First, consider minimizing OKScript by clicking the minimize button in the upper right. While this almost completely gets rid of the program in many cases it is too drastic.

The **Script | Set Panel Options** dialog lets you build panels without the file tabs at the bottom. This saves a little space. You can still switch between panels by hitting ctrl-tab.

This dialog also has a setting for Rollup Panels. When this option is enabled, a panel "rolls-up" into the title bar when not in use, and rolls back down when you click on it.

Finally, consider re-casting large button panels as menu panels. Menu panels take very little space plus you often benefit from a menu's inherent ability to structure information hierarchically.

Next - assorted topics

Tutorial page 8

Other Features

OKScript also includes iteration (looping), condition testing (IF-ELSE), integer formula, file and directory and clipboard capabilities. These are generally not too useful for OKbridge scripts (although I have seen them so used), but they can be helpful if you are developing scripts for other applications.

In addition to Search (and Replace) the script editor has two features that ease the development of large scripts - Keystroke Recording and Playback and Bookmarks. Use the record and playback feature whenever a systematic editing task is called for. For example let's say you have this script code:

```
MENU "ckbk|1d-1h-1s|2c=inv or obv game|2d=min w/o support"
MENU "ckbk|1d-1h-1s|2c=inv or obv game|2h=support"
MENU "ckbk|1d-1h-1s|2c=inv or obv game|2s=max w/o support"
MENU "ckbk|1d-1h-1s|2c=inv or obv game|2n=16-18,4252,4153 etc"
```

and you want to expand "inv or obv" to "invite or obvious". While this could be done with search and replace, it requires less typing to use the key playback function. To do this start by putting the cursor after the first "inv" and hit F4 - this starts recording. Now type "ite" and cursor to right beyond "obv". Now type "ious" and move the cursor down a line and back to follow the next "inv". Now hit F5 (F6, F7 or F8 are also available for this) which saves those keystrokes. Hit F5 again to play back what you just recorded.

Bookmarks are used to remember a particular cursor location while you move about in a large script file. To set a bookmark hit ctrl-F9 (or ctrl-F10, ctrl-F11 or ctrl-F12.) Now move the cursor away from that spot. Return to your bookmark by hitting the F9, the F-key *without* ctrl.

Specifying a Target Window

The WINDOW, BUTTON, MENU and RIGHT instructions contain a field that explicitly specifies a window to activate to receive keystrokes. There are two forms that this window specification can have. The first, and most common, is a string that is a part of the target window's title bar, e.g. here is a button that sends "the quick brown fox" to the window with "Notepad" in it's title bar:

```
BUTTON foo red Notepad the quick brown fox
```

The second approach is to activate the window using the window's handle, which can be determined using the %-ACTWIN% or %-TICWIN% predefined macros. This approach can be useful when there is more than one window active with the same name. To indicate that the window string is a handle number prefix it with the '#' character. Here is a BUTTON script that gets the handle of the Notepad when left-clicked and sends a string to it when right-clicked:

```
BUTTON handle red Notepad
- Notepad is now active.
- Get it's handle to the wHnd macro.
LET wHnd %-actwin%
- Activate Notepad via its handle.
RIGHT #%wHnd% the quickbrown fox
```

Updates and Reporting Bugs

Program information and the latest versions of OKScript are available at: **http://home1.gte.net/yweare**. (Should this site change you can find the new site by searching for "OKScript" with a search engine such as www.hotbot.com or www.metacrawler.com.) Please send comments and bug reports, with program version number (click **Help | About**), 'bitness' (16 or 32), operating system and any revelant scripts to: **yweare@gte.net**, or check the OKScript website for current information and message board.

Garbled Keystrokes

There have been reports of garbled results on fast machines when a lot of text is being sent to another application with one button/menu click. I am not clear about what is causing this (and haven't been able to replicate it) but have added a delay mechanism that might help. Preset delay values can be set from the Set Panel Options dialog. Custom delay values must be set in the OKSCRIPT.INI file which is in the OKScript folder. Open this file with Notepad or the OKScript editor and modify these lines to suit your needs:

```
[SYSTEM]
chardelaydefault=5
enterdelaydefault=700
```

Next resave the file, exit and restart OKScript. The values are in milliseconds. (Shown are the original default values.)