Push: ThinHelp's hypergraphics powerhouse

Push: an overview

What exactly does Push do?

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushicon.bmp;0/0/0/0/0/128/128}PUSH.EXE is "SHED for ThinHelp", a full-featured hypergraphics editor with everything you're likely to need in a hypergraphics editor except for actual painting tools. It's the most versatile and complete replacement for SHED.EXE ever released, and could very well take SHED's place for all your hypergraphic chores. Here's a brief list of its capabilities with links to the associated Help topics:

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Creation of SHED/image-map-style hypergraphic hotspots in Help bitmaps.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Fast two-step conversion of existing .SHG files to high-compression, resolution-independent ThinHelp embedded hypergraphics.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Editing of .SHG hotspots without the need for SHED.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Conversion of various standard bitmap formats to ThinHelp-compatible .BMPs and JPEGs.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Export of resolution-independent .SHG graphics...no more worries about scaling or end-user font sizes.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Clean, crisp scaling of screen shots and bitmaps for reduced-size display.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Addition of WMF metafile banners to embedded-window .BMPs and JPGs.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Precision control over hypergraphic hotspot dimensions with a built-in zoom function.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLYEL.BMP;0/0/0/0/192/192/192} Automatic import and update of existing SEG.EXE hypergraphics datafiles for quick conversion of JPANI/LZANI .SEG files to ThinHelp-compatible .SEGs

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

HelpDebug: troubleshooting your Help projects

{ewl THNHLP,THIN,!PUSHBMPS.LZH;HLPDEBUG.BMP;0/0/0/0/255/255/255}HelpDebug was originally developed for use with ThinHelp's own help system as a means of checking link integrity and insuring that some common authoring errors that the compiler doesn't report, specifically errors in embedded window statement syntax, were caught and fixed before the help system was released. It has evolved into the much more thorough tool you now have at your disposal for checking a wide range of helpfile errors that were previously uncheckable without extensive manual testing.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`hda')} About HelpDebug Lite

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`hdb')} How HelpDebug checks your Help's health

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`hdc')} HelpDebug's error reporting

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`hdd')} HelpDebug step by step

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`hde')} Using HelpDebug with multi-file Help systems

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`hdf')} What HelpDebug won't check

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

About HelpDebug Lite

The version of HelpDebug included with ThinHelp is a "lite" version of HelpDebug++, and as such it has some limitations and features not found in the full version of HelpDebug++.

- HelpDebug Lite is designed for 16 bit Windows Help projects only. 32 bit HCRTF.EXE-compiled helpfiles and Viewer titles are not supported. Error checking is possible with both of these file types insofar as they are compatible with 16 bit Help's file format and RTF tagging protocols, but this application of HelpDebug Lite is not supported.
- HelpDebug Lite is integrated into the Push hypergraphics editor. It is not available as a stand-alone utility with ThinHelp and must be launched by loading an .HPJ file into Push.
- HelpDebug++ supports saving and printing of error and project statistics reports. There is no option to save or print error reports in HelpDebug Lite.
- Custom third-party Help/Viewer DLLs and functions can be added to HelpDebug++'s error-checking capabilities via an extension interface. HelpDebug Lite will check syntax for custom macros, but it does not check for syntax errors in embedded window statements other than ThinHelp's own.

- HelpDebug++ will include an optional interface to ASETUP for automatic generation of self-extracting, auto-starting installation archives for effortless creation of setup packages for product updates, catalogs, stand-alone Help/Viewer projects and more.
- HelpDebug++ will support the multimedia embedding functions of Viewer 2.0, offering among other things the ability to error-check 256 color SHGs.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`IDH_HELPDEBUG')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

How HelpDebug checks your Help's health

HelpDebug Lite is activated automatically when you load your project's .HPJ file into Push. First it checks to make sure all needed source files are found where you have referenced them in the .HPJ file. Then it checks the [FILES] section for your RTF source documents.

Next it examines your source RTF files, character by character, and looks for anything amiss...macros that aren't authored properly, embedded window statements that don't make sense, bitmaps referenced to the wrong compressed graphics library, topics that aren't linked properly, and jump and popup macros that don't go to the correct topic (a critical factor when authoring with ThinHelp and something that the Help compiler won't check for you). All results are logged to its internal database.

Finally it displays its results window as a series of tables representing project statistics. A drop-down list is provided at the top of the window allowing you to select the type of data you want to see displayed.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`IDH_HELPDEBUG')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

HelpDebug's error reporting

Naturally the first thing you'll want to see is an error report, and ThinHelp's error reporting goes far beyond that of the Help compiler. It's an "instant bug list" for quickly repairing source file errors prior to compiling and getting error-free compiles on the first try without the bother of manually checking links and macros that the Help compiler doesn't check.

Here's a list of the types of errors which HelpDebug will report that the compiler won't catch. The green bullets indicate items which users of the ForeHelp interface will not have to worry about.

- invalid or incorrect interfile links
- invalid or incorrect JumpID and PopupID macros both within the current helpfile and in all helpfiles found in the project directory which the current helpfile might reference
- topic titles used in more than one topic
- missing parentheses at the end of a macro (especially useful for debugging complex nested macros)
- invalid macros, even macros-within-macros such as those used in IfThenElse() and ChangeButtonBinding() macros
- footnote references (#, $, K etc.) with more than one character, a common occurrence when migrating RTF source between versions of Word or between authoring tools
- illegal characters which Help cannot display; especially common with non-American versions of Word
- improper data types used in ThinHelp's custom macros and other custom API or DLL macros used in your project

- missing or incorrectly referenced bitmaps in ThinHelp embedded window statements
- syntax errors in ThinHelp embedded window macros (something non-ForeHelp users will find indispensable as they get used to ThinHelp's use of tildes instead of commas)
- incorrect or invalid display parameters in ThinHelp embedded window statements
- incorrect semicolon usage in ThinHelp embedded window statements

When the analysis is finished, you'll also have access to a number of other useful reports for helping you discover files that don't need inclusion, authoring errors in browse sequences and misplaced keywords. HelpDebug Lite is a vast improvement over the error-checking provided by the Help compiler and it's certain to save both time, effort and mental stress for any Help developer.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`IDH_HELPDEBUG')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

HelpDebug step by step

1. Activate HelpDebug to error-check your Help project by loading the .HPJ file into Push. Click **File** from Push's top menu bar and selecting **Read Help project (.HPJ) File...**. You will be presented with a Windows **File|Open** dialog box for selecting the .HPJ file.

2. In anywhere from a few seconds to a minute or so you'll see a report window pop up over top of Push with HelpDebug's findings. The first time you run HelpDebug you will probably find that its window dimensions are inappropriate for your screen resolution. (Resize the window to the maximum allowable screen width for better display. When the window is closed these dimensions will be saved as defaults and used for all subsequent executions of HelpDebug.)

3. HelpDebug's error report is always the first report which appears on the screen. Use this report as a guide to errors needing attention. You may need to repair your RTF and/or HPJ source and **Update** the report several times before you get an error-free analysis.

4. Before exiting HelpDebug and finishing your project, be sure to check the **Required files** list to insure that your distributed Help project includes all these files as part of its distribution. This list will also tell you if you have incorrectly referenced one or more embedded window statements. For example, if you authored your helpfile with all compressed graphics libraries included as Help baggage and you see one of your .LZH files listed as a required file, this indicates that you have forgotten to add a "**!**" to the LZH archivename parameter in one or more of your embedded window statements.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;IDEA.BMP;0/0/0/0/0/255/0}Any additional reports can be selected from the **List selection** drop-down list. Reports can also be sorted by topic or data type, or by file for projects with multiple RTF source files, by clicking **File**, **Topic** or the data type header on the header bar just beneath the **List data** caption, or by right-clicking within the data display area.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`IDH_HELPDEBUG')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Using HelpDebug with multi-file Help systems

If you're in charge of a large multi-file Help system, HelpDebug can provide you with a complete checklist of all improper links in all files in the system. Checking interfile link integrity in multi-file systems is a task that could previously take several days to complete with projects containing several thousand links.   HelpDebug takes care of it in just minutes.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;0/0/0/0/0/255/0} HelpDebug Lite will not check the integrity of compiled helpfiles.   In order to completely test the integrity of a multi-file Help system you must take the following two steps prior to testing each helpfile in the project:

1. Copy all needed helpfiles (and externally stored LZH files if necessary) to the root directory of the helpfile you wish to analyze, or insure that the needed files and extensions are referenced in WINHELP.INI or located within the Windows path.
2. Insure that all baggageable .LZH graphics libraries for this helpfile can be found at the locations specified in its .HPJ file.

Now load the .HPJ file into Push.

These precautions should insure a minimum of “false error reports” based on HelpDebug’s inability to find a component of the Help system which is referenced in the helpfile currently being analyzed.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`IDH_HELPDEBUG')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

What HelpDebug won’t check

HelpDebug Lite does an extremely thorough job of troubleshooting 16 bit helpfiles, but it is by no means perfect.   Here’s a list of some of the things it will *not* check for:

- incorrect placement of bitmaps in a topic
- hotspots or macros which jump to valid links which are not appropriate links
- incorrect versions of extension DLLs
- RTF formatting tags or WinHelp macros used exclusively in 32 bit helpfiles (HelpDebug Lite is not intended for use with 32 bit Help projects)
- popups which are incorrectly coded to jump to secondary windows (these are reported by the compiler if the popup link is located in the same helpfile as the anchor text)
- undocumented WinHelp macros.   Some macros, such as Command(), Generate() and JumpHash() are available for use in your Help projects and are supported by some authoring tools (e.g. Help Writer’s Assistant and HDK) but were never properly documented by Microsoft.   HelpDebug Lite does not recognize these macros and will report an error if they are used in your helpfiles.   These errors do not mean that your macros will not work, only that HelpDebug does not recognize the macro.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`IDH_HELPDEBUG')}

Push's project management features

Automatic, one-touch bug checking for Help projects

{ewl THNHLP,THIN,!PUSHBMPS.LZH;HLPDEBUG.BMP;0/0/0/0/255/255/255}The registered version of ThinHelp includes an integrated "lite" version of our HelpDebug utility for checking the integrity of your Help projects prior to compiling. HelpDebug includes its own online Help at this link. HelpDebug runs automatically each time you open your .HPJ file into Push.

Automatic project handling and macro syntax checking

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushicon.bmp;0/0/0/0/0/128/128}Push is integrated with HelpDebug and reads context IDs directly from your Help project. Simply load your .HPJ file into Push and you'll have a handy drop-down list of topic IDs to select from when you define new jump or popup hotspots. Any macros or window definitions you specify in Push hotspots will also be checked for accuracy.

File locations stored in .SEG data

{ewl THNHLP,THIN,!PUSHBMPS.LZH;brain.bmp;0/0/0/0/0/255/0}Push also remembers the locations of the files it processes. When you load a bitmap into Push and add hotspot information to it, the resulting information is saved in a separate file which contains both the hotspot information and the name and location of the source bitmap on your system. The next time you load a saved .SEG file, Push will automatically find and load the associated bitmap if it still exists at its former location.

The MRU (Most Recently Used) list

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushbutn.bmp;0/0/0/0/0/128/0}Push keeps a list of the nine most recently used .SHG or .SEG files in its **File** menu for easy access. Since Push is not intended as a graphics processing tool, but as a hypergraphics editor, it only keeps track of .SHGs and .SEGs, not bitmaps.

Preferences

{ewl THNHLP,THIN,!PUSHBMPS.LZH;magfold.bmp;0/0/0/0/0/255/0}In order to speed up the process of debugging your Help and loading topic ID information for use in Jump/Popup hotspots, Push can save the name and location of your current project, load it and automatically run HelpDebug whenever Push is launched. The **Preferences** item under the **File** menu calls up a dialog which retrieves and records the name and location of your Help project file.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

Preparing to work with Push

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushicon.bmp;0/0/0/0/0/128/128}PUSH.EXE looks and feels much like SHED.EXE, and does virtually everything all known versions of SHED can do except for creating visible hotspots. If you are familiar with SHED.EXE you should have little difficulty working with Push at a basic level.

Functional differences between SHED and Push

{ewl THNHLP,THIN,!PUSHBMPS.LZH;GEAR.BMP;0/0/0/0/0/255/0}The most important functional differences between SHED and Push from an authoring standpoint are that Push can save both true .SHG files compatible with Help and Viewer and also save the exact same hotspot data to ThinHelp's proprietary .SEG format which allows for superior compression performance in your compiled projects. The .SEG data is stored separately from the bitmap in its own file similar to the way World Wide Web image map generators save .MAP files. This allows you to add hotspots to 256 color .BMPs for display in all common versions of WINHELP.EXE and even offers the ability to assign hotspots to TrueColor JPEG bitmaps.

SHED, on the other hand, saves hotspot information in the same file as bitmap itself and is limited to the inefficient RLE and/or LZ77 compression algorithms used by the Help and Viewer compilers.

Push also allows forms of WinHelp-compatible .SHGs which SHED can't handle. Help to RTF will correctly extract .SHG files from Windows 95 32 bit helpfiles which have been compressed using both RLE and LZ77 compression, and Push handles these files correctly for processing and recompiling into 32 bit Help. SHED, on the other hand, can't deal with these files. It expects RLE- or LZ77-compressed files only.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;CAUTION.BMP;0/0/0/0/0/255/0}Push is not capable of generating metafile (.WMF) based SHG files. Instead Push will convert any metafile loaded into it into a bitmap. This usually degrades the display quality of the bitmap and increases its file size unless the source .WMF was a "pseudo-metafile" generated by Paint Shop Pro from a source bitmap.ing the file size. We recommend using SHED.EXE to process .WMF-based SHGs.

The only other significant difference between Push and SHED is that Push does not support visible hotspot boundaries. These will have to be drawn in manually using other graphics software if they are required. They are so seldom used or needed in helpfiles (they are not even available in HTML authoring) that we did not add this feature to this version of Push.

There are many more subtle differences in the features provided by SHED and Push. As you progress through your first few Push hypergraphics you'll discover how these features work and where to apply them in your authoring. You'll find Push provides a lot more "room to grow" than SHED does.

Working with the dropdown topic ID list

{ewl THNHLP,THIN,!PUSHBMPS.LZH;MAP.BMP;0/0/0/0/0/255/0}The most significant aspect of Push's behavior in comparison to SHED is the way it wires into your Help project. Because Push is designed to work with ThinHelp, it doesn't completely "mesh" with Help or Viewer (i.e. you can't simply generate Push's proprietary .SEG file format, reference it in your project as *{bmc filename.seg}* and expect it to work). On the other hand, it can extract and use data from your helpfile in ways that SHED can't.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;NOTE.BMP;0/0/0/0/0/255/0}Push relies on its own integrated version of HelpDebug Lite to compile a list of all topic IDs from your source RTF documents, and it allows you to either select these from a dropdown list or define new strings for topics in other helpfiles or topics which you have not yet authored. Push will issue a warning when no topic ID matching the one specified in the hotspot attributes was found by HelpDebug, or if HelpDebug has not been applied in the current editing session. These warnings will appear at least twice when you create popup and jump hotspots, but they have no effect on Push’s ability to save the file and do not indicate problems in your compiled project, unless of course the hotspot you specified was entered incorrectly.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

Push hypergraphics step by step

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Starting Push

1. Start PUSH.EXE. If you want to avoid a lot of alerts and save a little time searching out topic ID strings while adding hotspots, debug your Help project before editing any new hypergraphics. You can activate HelpDebug at any time during an editing session by loading the project's .HPJ file into Push. Click **File** and select **Read Help Project (.HPJ) File...** to automatically start HelpDebug and compile a list of topic IDs for the the dropdown list in the **Hotspot Attributes** dialog. (For more information on HelpDebug, see the section devoted to this utility.) If you intend to work on the same Help project for several sessions with Push, use the **Preferences** entry to select its HPJ file for automatic loading and debugging at startup.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushsteps')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Loading bitmaps and hypergraphics

2. There are several load options available for selecting graphics datafiles for editing in Push. Here's how to load each type:

   a) If you want to work with existing .SHG files, load the .SHG into Push by clicking **File** from the top menu bar and selecting **Open SEG/SHG...**. Note that Push can only handle .SHG files generated from .BMP files. It will not process .SHGs generated from .WMF metafiles.

   b) If you want to work with a bitmap on your hard disk, load it into Push by clicking **File** and selecting **Open Picture File...**. PUSH.EXE will import WMF, DXF, BMP, TIF, GIF, PCX and JPG file formats.

c) If you want to use a bitmap which you have been editing in another graphics program, you can copy this bitmap directly into Push's editing window. Here's how:

1. Select the bitmap from the graphics editor (**Select All** is the most commonly used menu item for this function) and copy it to the clipboard.

2. Task-switch back to Push.

3. Click **Edit** from the top menu bar of Push and select **Paste Bitmap**. This option will only be available if there is actual bitmap data on the clipboard which can be pasted.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushsteps')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Editing bitmaps and hypergraphics

3. Apply any needed filtering or scaling to the bitmap before adding hotspots. This will usually insure the most efficient results. (See the associated Help topics for more information on these features of Push.) Scaling can be applied after hotspots are defined if you like. Push will ask whether you want your hotspots automatically reproportioned when the rescaling operation is complete.

4. Create your hotspots just as you would with SHED.EXE or an image map tool by clicking and dragging the mouse to create rectangular hotspots. These hotspots can be moved and resized by clicking anywhere inside the hotspot and dragging the entire hotspot to its new location. The status bar at the bottom of Push's window will provide information on hotspot coordinates and attributes.

   If you want more precise control over the size or location of the hotspot, use Push's zoom function (under the **View** menu) to zoom in on the graphic and tweak the hotspot size.

5. a) Add hotspot ID information to your hotspots by *double*-clicking inside the hotspot. A **Hotspot Attributes** box will pop up which functions almost identically to SHED's, with the exception of a handy drop-down list of jumps and macros available in the **Binding** section. This list can be used to insure correct spelling of any jump or popup ID string.

   {ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLRED.BMP;0/0/0/0/192/192/192} **Note:** You are likely to encounter difficulties if you attempt to create a new hotspot directly on the surface of a bitmap smaller than 14x14 pixels in size. Generally speaking it is more sensible to attach macros to these bitmaps in your RTF source than to add hotspots to them using Push.

   b) Add metafile overlays to hotspots by pasting them directly from the clipboard using either **Ctrl+V** or the **Paste Metafile** entry under the **Edit** menu. A hotspot can have both a metafile overlay and a hotspot attribute, but you can only have one metafile overlay per hotspot.

   {ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLRED.BMP;0/0/0/0/192/192/192} **Note:** You must plan your metafile overlays carefully since Push doesn't allow the same flexibility with overlapping hotspots that SHED allows. Use two or more smaller, identically-attributed rectangular hotspots to achieve the same effect as an overlapped hotspot. While this takes a little more effort than working with SHED overlapping hotspots, you'll never have to worry about hotspot order again, and we recommend avoiding overlap with SHED too for this very reason.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushsteps')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Saving bitmaps and hypergraphics

6. Save your graphic and/or hotspot information using the following guidelines:
   a) If you have been working on an .SHG file or an existing .SEG file on your hard disk, clicking **File** and selecting **Save** will save the edited .SHG or .SEG. .SEGs can be saved as .SHG files and vice versa. Saved .SHG files will be fully compatible with .SHGs generated with all known versions of SHED, SHED2, MVSHED and Enhanced SHED. If modified .SHGs are to be saved as .SEG files for an additional reduction in compiled file size, make sure you also save the SHG's source bitmap as a .BMP prior to closing Push.

   b) If you have been working on a new hypergraphic, you will need to save it as either a .SEG file or as a SHED-compatible .SHG file. Click **File** and select **Save**. When the **File|Save** dialog pops up, save and you will be prompted to give a name to your new .SEG file. **.SEG file naming convention follows the name of the bitmap file with which it will be used,** so if your bitmap is named MYICON.BMP, the .SEG file should be named MYICON.SEG.

   c) If you are working with a non-BMP or non-JPEG bitmap, *or* if you have modified the image using Push's scaling or matrix filters, you will also need to save the bitmap in a ThinHelp compatible format. ThinHelp compatible bitmaps should be saved in either BMP or JPG format. **It is strongly recommended that modified TrueColor bitmaps designed to be displayed as JPEG graphics be saved twice...**once as a .BMP "master" and *at least* once as a JPG for inclusion in your project. **(**See the section on exporting for more information on saving JPEGs.)

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushsteps')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Referencing Push-generated bitmaps and hypergraphics in source RTF files

7. Insure that your modified graphics, .SHGs and/or .SEGs are correctly added to your project using the following guidelines:
   a) If you edited and saved an .SHG file, you can either treat this as a normal .SHG file to be referenced in the usual fashion, or you can display it as a ThinHelp embedded window.

      Keep in mind that 256 color SHGs are only recommended for use with 32 bit helpfiles, Viewer projects, or projects developed on platforms which support Microsoft's SHG file format. If you wish to display 256 color .SHGs in 16 bit helpfiles we strongly recommend displaying them as ThinHelp embedded windows for full compatibility with Windows NT and Windows 3.1/3.11.

      Also be aware that the end user may encounter scaling problems when attempting to print normal .SHG files. If you wish to insure accurate printing of the .SHG's bitmap data, we recommend displaying any .SHGs which originated as .BMPs as ThinHelp embedded windows.

   b) If you created or edited a .SEG file, this file must be added to same LZH-compressed archive that contains the bitmap with which it is associated. If this bitmap is a JPEG graphic, then the .JPG file and its associated .SEG file *must* be included in the *same* LZH-compressed graphics library. A JPEG and a .SEG file cannot be added to baggage without LZH compression...the "no-compression-needed" feature of ThinHelp is reserved exclusively for non-hotspotted JPEGs.

8. If you are creating a ThinHelp embedded window, reference the bitmap file in your RTF source using the appropriate embedded window syntax. Instead of using a .BMP or .JPG extension in the file's name, it should be referenced with no extension at all. This will tell ThinHelp to load both the bitmap *and* the associated .SEG file when it displays the bitmap.

   For example, if you were embedding MYPIC.JPG and MYPIC.SEG in a helpfile, and the bitmap was stored as a compressed graphic in BITMAPS.LZH, you would reference it as:

   {ew*x* THNHLP,THIN,!BITMAPS.LZH;MYPIC}

   ...if BITMAPS.LZH is to be stored *in*side the helpfile as baggage, or...

   {ew*x* THNHLP,THIN,./BITMAPS.LZH;MYPIC}

   ...if BITMAPS.LZH is to be stored separately on disk in the helpfile's directory.

9. Compile and test. If you encounter problems which were not covered in this topic, jump to the troubleshooting section for more help.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;CAUTION.BMP;0/0/0/0/0/255/0}Be careful not to use .SEG extensions in your embedded window statements. ThinHelp can't handle a hotspotted bitmap referenced as BITMAP.SEG.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushsteps')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Notes, tips and issues

{ewc THNHLP,THIN,!PUSHBMPS.LZH;CHECKMRK.BMP;0/0/0/0/255/255/255} Be sure to use the appropriate HCRTF.EXE-compatible abbreviated macro names when authoring .SEG and .SHG hotspot macros for 32 bit helpfiles. HCRTF.EXE won't automatically convert JumpID to JI, for example, when it sees this macro in the embedded window statement. Most 16 bit Help macro syntax will function properly when compiled in a 32 bit project but some macros will not.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;CHECKMRK.BMP;0/0/0/0/255/255/255} Any graphics software capable of handling bitmaps as metafiles can be used to generate metafile-format graphics usable by PUSH.EXE. JASC Inc.'s Paint Shop Pro is a shareware program which will allow you to open virtually any type of bitmap, cut or copy all or part of it to the clipboard, and paste it into a Push hotspot as a stretchable metafile overlay. Try it...the results range from satisfying to disgusting, and it's another unique technique to add to your Help authoring toolkit.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;CHECKMRK.BMP;0/0/0/0/255/255/255} The file format for .SEG files is documented in SEG.H if you require it for a particular application or extension to your project.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushsteps')}

{ewl THNHLP,THIN,!
PUSHBMPS.LZH;CAUTION.BMP;0/0/0/0/0/2
55/0}If you decide to use JumpID() and/or
PopupID() macros in Push hotspots in place
of standard **Jump**/**Popup** hotspots, shorten
the references to their abbreviated forms (**JI**
or **PI**) prior to saving the .SEG file.   This will
prevent problems with compiled 32 bit
helpfiles which rely on abbreviated versions of
Help’s macro names.

Scaling bitmaps

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushicon.bmp;0/0/0/0/0/128/128}Push offers intelligent scaling of screenshots and other difficult-to-display bitmaps so that they can be shown within a desired Help window size regardless of the end user's screen resolution. It's a quick solution to the problem of displaying large screenshots in a minimum of Help window space, and ideal for quickly creating high-quality "thumbnails" of large graphics, especially JPEGs, for use as "pick-and-click" menu bitmaps. It is *not* suited to reducing the sizes of all your bitmaps. Size reduction nearly always produces a degree of image quality degradation regardless of the process used.

Steps

1. Select a bitmap you wish to rescale.
2. Start PUSH.EXE and load the bitmap into Push from **File|Open Picture File...**. If it's an .SHG file you will need to select **File|Open SEG/SHG...**. You may load non-BMP, non-JPEG, non-SHG graphics but they must be a Push-compatible file format.
3. Select **Edit** from the top menu bar and click one of the three **Scale** ratios.

   1:2 scales the bitmap to one-half its width and one-half its height, or *one-quarter* of its original area.

   2:3 scales to two-thirds width and two-thirds height, or a little less than one-half of the original area.

   3:4 scales to three-quarters by three-quarters of the original dimensions.
4. If you have a 256 color display, what you see on your screen after the reduction may not match the actual appearance of the bitmap in your helpfile. Push converts bitmaps to 16 million color TrueColor bitmaps in the scaling process. In order to preview the scaled bitmap, select **Edit** again and click **Reduce to 236 colors**. When the reduction is complete, what you see on your screen will be an accurate rendering of the rescaled graphic. Although scaled bitmaps generally look best when preserved at 16 million colors, most reduced screen shots and line drawings, as well as thumbnail-sized TrueColor bitmaps, will produce quite acceptable display results when reduced to 256 colors.
5. In most cases, reducing a rescaled bitmap to 16 colors will result in an unusable graphic. Before experimenting with reduction to 16 colors in a scaled graphic, it is recommended that you save your work. Select **File** and click **Save Picture As...**. You can select any supported format for saving the graphic. (At 256 color depth, BMP is usually the preferred format.

   {ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLRED.BMP;0/0/0/0/192/192/192} **Note:** If there is a chance that further processing of the bitmap may be required, *do not* save it as a JPEG. *Always* preserve a .BMP copy of any bitmap you wish to include in your project as a .JPG. Use the TrueColor .BMP, not the .JPG, as the base bitmap for all future editing. (See the section on exporting for information on saving JPEGs.)
6. In many cases, particularly with photorealistic renderings, reducing a scaled bitmap to 256 colors for saving as a .BMP will result in a *negative* effect on file size. You may wish to try saving both a .JPG and a .BMP to see which produces the smallest file size when added to your helpfile's baggage. Sometimes you have to actually add a .BMP to an LZH graphics library and tally the difference in file sizes to determine which method offers the best overall compiled size.

```
{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}
```

Push's matrix filters

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushicon.bmp;0/0/0/0/0/128/128}In addition to its scaling algorithms, Push includes filters for improving the esthetic appearance of rescaled images and TrueColor bitmaps. These include a Sharpen, Soften, Brighten and Darken. They should behave exactly as you'd expect based on their names, and if you've dabbled with programs such as Paint Shop Pro, Hijaak, Photoshop or PhotoStyler, you already have a good idea what to expect from them.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`filtguid')} General guidelines

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`filtbrit')} Brighten and darken

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`filtsoft')} Soften

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`filtsharp')} Sharpen

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

General guidelines

The four matrix filters in Push are designed exclusively for use with TrueColor images. If you do not load a TrueColor image into Push, then the only time you will have access to these filters is after you scale a bitmap. (Scaling always results in a TrueColor image.)

Multiple passes of these filters on the same bitmap are not merely permissible, but often recommended for optimum results. You will seldom require more than two passes of any filter to achieve optimum appearance.

Reduction to 256 colors and save-as-BMP are highly recommended for screen shots and line drawings which have been rescaled and then filtered. Generally speaking, screen shots and line drawings saved as 256 color LZH-compressed .BMPs will result in smaller compiled projects than the same bitmaps saved as .JPG graphics. Experiment with the available quality options when saving as .JPG to determine the best ratio of quality to file size.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushfilter')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Brighten and darken

The brightening and darkening filters are intended for use with complex (usually photorealistic) images which lose critical details when scaled. If possible the brightening or darkening should be done *before* scaling so that the best possible contrast balance is passed along to the reduced graphic. The resulting effect varies from image to image, however, and the actual results for any given image can only be determined through experimentation.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushfilter')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Sharpen

The sharpening filter will likely become your most-used Push matrix filter. It should never be applied to a bitmap prior to scaling unless this bitmap is not going to be scaled down, and it should never be needed with unscaled screen shots or line drawings.

If you only want a single size reduction of 3:4 or 2:3, then you should apply this filter *after* scaling the image. If you need a reduction of 1:2, or if you intend to use multiple applications of the scaling filters, you may want to try reducing by 3:4, applying a single pass of the sharpening filter, then reducing by 3:4 again and sharpening a second time (and perhaps a third) after that.

Effective use of the sharpening filter requires some experimentation. When scaling screen shots, you will usually (but not always) achieve the best *esthetic* appearance with a single pass of the sharpening filter and the most *accurate* appearance with two passes. A second pass of the sharpening filter on a reduced bitmap will usually result in speckling, whereas one pass, a further reduction, and then another pass will keep speckling to a minimum. (Many third-party graphics processors include de-speckling filters if additional fine-tuning is required on the bitmap. Despeckling will generally, but not always, reduce the bitmap's space requirement when the bitmap is added to an LZH archive.)

Sharpening can be applied to highly detailed thumbnails of reduced photorealistic graphics to enhance subject detail in the reduced graphic. This nearly always results in a degree of distortion, but the distortion is generally pleasing to the eye and often makes the thumbnail's subject matter more intelligible.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushfilter')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Soften

This filter will also require experimentation. It produces a blurring effect in your graphic which is usually much less drastic than **Blur** filters available in many graphics processing utilities.

One trick used by application-oriented Help experts for making crisp reductions of screen shots is to soften a screen shot prior to any serious downscaling. Push's scaling filters are optimized to produce clear results when reducing your graphics, so this may not be desirable unless the default font size of the original screen shot is 12-point bold or larger.

On the other hand, if you have a photorealistic graphic with a lot of high-contrast detail, such as sunlight through trees, sun on water or a sunlit snowy landscape, you may find that a pass of the softening filter prior to scaling will enhance the overall appearance of the scaled graphic.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushfilter')}

Push scaling and filter examples

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushicon.bmp;0/0/0/0/0/128/128}This topic presents two sample images and demonstrates the results that can be obtained using various combinations of scaling and filtering.

Example 1: a thumbnailed screen shot

This first example shows a Notepad screen shot without scaling, and the same screen shot after a 1:2 reduction in size, *two* passes of the sharpening filter, and a color reduction to 256 colors optimized. The window borders in the first reduction are a little speckled, but the text is still readable.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;NOTEPAD1.BMP;0/0/0/0/0/255/0}

Now a second reduction is applied. The image on the left was scaled down from the original Notepad screen shot by a ratio of 1:2, reduced *again* by 1:2, then sharpened. The image on the right was reduced, then sharpened twice, then reduced a second time and sharpened twice again. The difference is subtle in this example, but it could be more striking with other screen shots or less severe reductions in size. You'll have to experiment to achieve the best results with a given type of image. Use the **Reload** option under the **File** menu to reload the original graphic and start over if you don't like the results of your experiments.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;NPRSHARP.BMP} {ewc THNHLP,THIN,!PUSHBMPS.LZH;NPRSHARP.BMP}

Example 2: a "difficult" photorealistic image

Now let's see what happens with a TrueColor photorealistic graphic. (These examples have been reduced to 256 colors for demonstration purposes.) If you have a 256 color display, scroll the screen so that the Notepad bitmaps are no longer visible to prevent palette conflict distortion.

The next photo was poorly shot. We know that Help authors very often don't have top-quality graphics to work with, which is why a poorly-rendered image was selected here as an example.

{ewr THNHLP,THIN,!PUSHBMPS.LZH;PHOTO.BMP;0/0/0/0/255/255/255}When reduced by 1:2 *twice* to create the top right thumbnail image, the result is a dark, difficult-to-understand image. The image at the bottom right represents the same thumbnail after being **Sharpen**ed twice, then **Brighten**ed twice. You'll probably find that the bread and cheese are much more readily identifiable in the lower right image than in the one in upper right.

This example demonstrates how important it is to have quality *original* graphics to work with. Although some enhancement was achieved using Push's filters, neither thumbnail looks like the appetizing feast it was intended to represent, and there isn't a lot you can do about this without heavily editing the original bitmap. The wine glass, for example, is barely identifiable in the original graphic and disappears completely in the reduction. It will require a graphic artist's hand to enhance it properly.

A note about conflicting 256 color palettes

{ewc THNHLP,THIN,!PUSHBMPS.LZH;NOTE.BMP;0/0/0/0/0/255/0} The scaling and filtering examples in this topic produced scaled bitmaps with conflicting 256 color palettes. If the three Notepad bitmaps or the three bread-and-cheese shots had been displayed without modification, significant color distortion would be visible on 256 color displays. We dealt with this problem by assigning a shared palette to each set of images so that all the Notepad screenshots used the same palette and all three photo images used another shared palette.

If you intend to display both the original image and its thumbnail on the screen at once, or multiple thumbnails in the same area of the topic window, you should create a matched, optimized 256 color palette for use by all thumbnails so that color distortion in 256 color display mode will be kept to a minimum.   256 colors is now the standard for Windows video display. You should never assume that your user has HiColor or TrueColor capability, or the knowledge or desire to switch to this mode.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;HLP-BUTT.BMP;0/0/2/0;;JI(`techdata.hlp>main'~`ID_PALETTE')}For more information on palette matching and palette-related issues with ThinHelp, browse the technical advice topic in the main ThinHelp helpfile linked to this button.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

Converting bitmaps

{ewl THNHLP,THIN,!PUSHBMPS.LZH;pushicon.bmp;0/0/0/0/0/128/128}In addition to being a hypergraphics creator, Push is also a versatile import/export filter and a simple graphics converter. Stand-alone graphics processing software is much better suited to the task of converting multiple bitmaps, but Push can certainly handle most of your single-bitmap conversion needs.

To convert a bitmap from one format to another using Push, simply click **File**, select **Open Picture File...** and choose from the available file formats listed under **Files of type:** in the **Graphic Import** dialog box. Then save your bitmap by clicking **Save Picture As...** and selecting the type of file you want to save from the dialog box at the bottom of the window.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`conva')} Accepted formats for import

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`convb')} Accepted formats for export

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`convc')} Resolution-independent export

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`convd')} Pasting graphics from the clipboard

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`conve')} Color depth reduction

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Accepted graphics formats for import

By default, Push will import SHED's SHG-format hypergraphics, TIFF-format graphics, PCX, Windows standard BMP bitmaps, DXF (AutoCAD) format vector graphics, GIF, JPEG and WMF metafiles. The 16 bit version of Push can make use of any additional MS Graphic Import Filters you may have on your system for use with other applications, which may give you import facility for dozens of different formats.

Note that PUSH can also import RLE-compressed .BMP files, but ThinHelp's DLLs won't be able to display them. You might want to load the .BMP and save it without alterations just to make sure. The load-and-save will strip any RLE compression that might have been applied to the bitmap.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;BULLRED.BMP;0/0/0/0/192/192/192} **Not all MS Graphic Import Filters will produce acceptable results expected when importing, exporting and rescaling. Metafile-based SHG, WMF, CorelDraw and AutoCAD, and other vector-based graphics are likely to suffer from noticeable degradation in the conversion process. Try to stick with bitmaps (GIF, PCX, BMP, etc.) where possible as source files when converting from one format to another with Push.**

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushconv')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Accepted graphics formats for export (convert-and-save)

Valid formats for saving with both 16 bit and 32 bit versions of Push include SHG (Microsoft Segmented Hypergraphic format), WMF (Windows metafile format), BMP (Windows-compatible bitmap), JPG (JPEG-compressed), and TIF (Tagged Image Format).

SHG format export:

The file is saved as a Microsoft-compatible BMP-based .SHG graphic without resolution information. The removal of resolution information from the .SHG insures accurate display at all screen resolutions in compiled Help projects but may result in scaling distortion when printed if the .SHG is not displayed as an embedded window.

BMP format export:

No additional options are available. The file is saved as a standard Windows .BMP.

WMF format export:

No additional options are available. The file is saved as a standard Windows .WMF metafile. Keep in mind that WMFs are not just a different format, but an *entirely different type of graphic.* Exporting bitmaps from Push as metafiles is not recommended.

TIFF format export:

{ewr THNHLP,THIN,!PUSHBMPS.LZH;TIFSAV2.BMP;0/0/0/0/0/255/0}Four different options are available for export as .TIF files as shown in the diagram at right, but please keep in mind that none of these options will produce a bitmap which ThinHelp can display in your projects. This option should be reserved for saving TrueColor bitmaps for cross-platform compatibility in environments where you need to work with graphic artists or printers who require .TIF format graphics from you, or for saving FAX-compatible .TIF graphics.

JPEG is a lossy form of graphic compression. The lossless compression offered by Push's TIFF export option allows you to save a high-quality, compressed TrueColor bitmap when further editing of a photorealistic image may be required.

JPEG format export

{ewr THNHLP,THIN,!PUSHBMPS.LZH;JPEGSAV2.BMP;0/0/0/0/0/255/0}This diagram shows the five options available when you select JPEG as the filetype for saving your bitmap. Each option provides increasingly better display quality in the saved graphic at the expense of increased filesize. Experiment with individual bitmaps to see which level of output quality produces the best balance of reduced file size to bitmap display quality.

If you do not have a bitmap viewer, your Web browser can double as a bitmap preview application for browsing JPEGs, or you can simply reload the saved JPEG into Push. Save-as-JPG should only be used for bitmaps you are certain you want to have displayed at color resolutions higher than 256 colors.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushconv')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Resolution-independent export...no more MRBC blues

The compatibility and bitmap-handling enhancements provided by Microsoft's multi-resolution bitmap creator (MRBC.EXE), MediaView SHED and Stephen Jenkins' SHGREZ modifying tool are no longer needed with the introduction of Push.

Push exports resolution-independent SHGs for accurate, no-surprises display as normal Help bitmaps and adds the advantage of perfect printability if displayed as ThinHelp embedded windows. The scaling distortion which you may have observed in some helpfiles need never occur with Push. Simply load and save your SHGs with Push to insure proper scaling at all resolutions, and display it as a ThinHelp embedded window to insure accurate printing.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushconv')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Pasting graphics from the clipboard

Any bitmap file format not compatible with Push's supported import filters can be pasted from the clipboard directly into PUSH.EXE and saved in any of Push's supported file formats. Unfortunately, not all graphics formats are paste-compatible. In order to be pasted into Push, the bitmap must be copyable as a standard Windows DIB (device-independent bitmap) or DDB (device-*dependent* bitmap) format. Most graphics programs have this capability, but some do not.

Some bitmap editors, Paint Shop Pro for example, may also attempt to copy Windows metafile (WMF) format information to the clipboard. This type of data can only be pasted into hotspots as a metafile overlay...it cannot be used as a source bitmap to which hotspots can be added. This task still requires the use of SHED.EXE.

When pasting bitmaps from the clipboard you must use the **Paste Bitmap** item from the **Edit** drop-down menu or the **Shift+Insert** hotkey combination. (**Ctrl+V** pasting is reserved for pasting metafiles into hotspots or pasting context strings or macros in the **Hotspot attributes** dialog.)

When copying non-JPEG high-resolution graphic data for saving as JPEG data it is *strongly* recommended that you switch your display mode to HiColor, or -- if possible -- TrueColor (16 million color depth) prior to the copy-and-paste operation. If you do not switch to a high resolution display mode, any bitmap data copied to the clipboard will be limited to the color depth available on your system at the time of the copy operation.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushconv')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Color depth reduction

If you select a color reduction from the **Edit** menu, Push will perform color reduction on TrueColor and HiColor bitmaps and JPGs. The only two reduction options available from the **Edit** menu are 16 and 236 colors. Reduction to 236 colors is recommended prior to saving scaled screen shots. If the bitmap was originally a 16 million color bitmap, or the use of scaling filters resulted in a 16 million color bitmap, this format can be saved as well, but bitmaps with 16 million color depth cannot be used as ThinHelp embedded windows unless they are either converted to JPEG graphics or reduced to 256 colors.

Default palette

There are two options for 236 color reduction: **Reduce using Default palette** and **Reduce using Optimized palette**. If **Default palette** is selected, the saved bitmap will appear in your helpfile exactly as it appears in Push's editing window. It will use the predefined standard 256 color VGA palette.

Optimized palette

Reducing color depth to the colors defined in the default palette will usually result in a vastly inferior image to the same image reduced to 236 colors optimized.   The optimization process will automatically update the bitmap display in Push’s editing window, and what you see after the color reduction is precisely what you will get when using the graphic in your helpfile...provided, of course, that there are no palette conflicts between this bitmap and any other bitmaps in the same topic, and provided you are running Push on a system capable of 256 color display.

{ewl THNHLP,THIN,!PUSHBMPS;NOTE.BMP;0/0/0/0/0/255/0}Push does not permit you to increase the color depth of your graphics except when scaling.   Color depth reduction is a one-way process.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushconv')}

Working with metafiles in Push

{ewl THNHLP,THIN,!PUSHBMPS.LZH;HLP-BUTT.BMP;0/0/2/0;;JI(`thinhelp.hlp>main'~`ID_METAFILE')}Push allows you to place metafile graphics directly over top of embedded window bitmaps displayed in your projects. It offers this capability with both .BMPs and .JPGs destined for use in ThinHelp-enhanced helpfiles. These metafiles cannot be saved as separate graphics files from Push *except* as part of ThinHelp-specific .SEG files. For more information about metafile overlays and how they are used with ThinHelp, see the introductory topic linked to this Help button before continuing.

Creating metafiles

Metafiles are created with vector-based drawing tools such as CorelDraw, Microsoft Draw, Serif DrawPlus or our own FormPad. However, any graphics software capable of handling bitmaps as metafiles can be used to generate metafile-format graphics usable by Push.

Adding metafiles to ThinHelp bitmaps

1. Load the bitmap into PUSH.EXE using any of the accepted methods.
2. Define a hotspot area of any size using the mouse cursor in the same fashion as you would with SHED.EXE or an image map generator.
3. Switch to your graphics software. Select the vector graphics object (or use **Select All** from MS Draw to select a graphic composed of multiple objects) and **Copy** it to the clipboard.
4. Switch back to Push and either use **Ctrl+V** from the keyboard or **Paste Metafile** from the **Edit** menu. If the graphics software truly supports metafile cut-and-copy, your graphic should appear in the Push hotspot.
5. Resize the hotspot to the appropriate dimensions. The corner regions of the hotspot will change to “handles” when you move your mouse cursor over them. If you find that parts of your graphic have “disappeared” off the bottom or right edge of your hotspot, simply resize the hotspot until the entire graphic is visible.
6. Move the reshaped hotspot to the desired position over your graphic.
7. If you want to add a jump, popup or Help/Viewer macro to your hotspot, double-click the hotspot to call up the **Hotspot Attributes** dialog. This dialog offers identical functionality to the **Attributes** dialog in SHED.EXE with one exception: it provides for invisible hotspot boundaries only.
8. Save the .SEG file.
9. See THINHELP.HLP’s section on working with metafile overlays for help with including the metafile with your project.

Notes, tips and issues

{ewc THNHLP,THIN,!PUSHBMPS.LZH;CHECKMRK.BMP;0/0/0/0/255/255/255} For some interesting special effects, try copying bitmap graphics into Paint Shop Pro and pasting them into Push hotspots. Paint Shop’s unusual design uses a default which copies metafile information to the clipboard even if the source image is not a vector graphic, meaning that you can paste icons, small photos and other types of non-vector-based images into Push hotspots. You won’t get the transparency effects of true vector graphics, however, and you’ll have to do some careful stretching and sizing of the hotspot to avoid distortion of the overlaid image.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

Troubleshooting Push

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptruba')} Message: Can't find Topic ID in help project's MAP

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrubb')} SHG hypergraphics look strange in large font display modes or won't print properly

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrubc')} Scaled graphics look poor

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrubc2')} Saved JPEG graphics are of poor quality

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrubd')} Filters won't work

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrube')} Push complains "hotspot misses ID information" and won't save

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrubf')} Scaling or filtering went "one step too far"

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrubh')} SEG hotspots won't execute the desired action in the compiled project

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptrubh2')} A .BMP file that looks fine in Push won't show up in the compiled Help or Viewer title

{ewc THNHLP,THIN,!PUSHBMPS.LZH;DOCUMENT.BMP;0/0/0/0/0/255/0;;JI(`'~`ptother')} Other problems

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Message: Can't find Topic ID in Help project's MAP

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}This message appears when you define a jump or popup link which is not associated with a topic which HelpDebug was able to find when it analyzed your project. This message also appears when you try to enter a jump or popup hotspot and you have not yet loaded your .HPJ file for HelpDebug's analysis. If the hotspot ID or macro information is entered correctly, the warning can be ignored.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

SHG hypergraphics look strange in large font display modes or won’t print properly

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}The .SHG was probably never loaded and saved with Push.   PUSH.EXE formats your saved .SHG file in a manner that insures that its proportions are never altered when it is displayed on systems with screen resolutions different from yours.

Printing may be a problem with some bitmaps depending on how they were generated.   If loading and saving an .SHG file with Push does not solve your problem, you may need to format the .SHG as a ThinHelp embedded window.   .SHG files saved by Push and referenced as standard Help .SHGs may suffer scaling problems if printed.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Scaled graphics look poor

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}Rescaled graphics won’t always produce the results you want.   Crisp scaling is almost a “holy grail” for context-sensitive help specialists.   Push will produce better results than many commercial products, but it is by no means perfect.   There are times when you simply won’t get the kind of quality that you or your client expect.

If you expect your scaled graphics to look as crisp and perfect as the downsized screenshots you’ve seen in some of Microsoft’s helpfiles, you will be disappointed no matter what tool you use.   Microsoft uses a proprietary process to create their screen shots.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;CAUTION.BMP;0/0/0/0/0/255/0}One trap to avoid when using Push’s scaling filters is to *not* reduce color depth until your bitmap has been scaled down to the size you want.   Each time you reduce color depth after scaling, you lose picture quality.   Wait until the final scaling operation has been completed before reducing the image’s color depth to insure the highest possible quality in your resulting bitmap.

{ewl THNHLP,THIN,!PUSHBMPS.LZH;IDEA.BMP;0/0/0/0/0/255/0}You might also try applying Push’s sharpening filter a second time to the same screen shot.   Very often two application of the sharpening filter instead of one will result in a much crisper appearance, but this often comes at the expense of a little speckling, especially with screen shots.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Saved JPEG graphics look poor

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}Unlike most Windows graphics converters capable of JPEG export, Push makes use of the full power of its DaVinci export filters to offer you a range of export options.

{ewr THNHLP,THIN,!PUSHBMPS.LZH;JPEGSAV2.BMP;0/0/0/0/0/255/0}This diagram shows the five options available when you select JPEG as the filetype for saving your bitmap.   Each option, from 10 to 80, provides better display quality in the saved graphic, and you’ll have to experiment with individual bitmaps to see which option produces the best balance of reduced file size to bitmap display quality by viewing save JPEGs in your browser of choice.   If you do not have a bitmap viewer, your Web browser can double as a preview application for browsing

JPEGs, or you can simply reload the saved JPEG into Push.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Filters won't work

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}Push's matrix filters function only when applied to 16 million color TrueColor graphics. They cannot be applied to 256 color graphics. You will either need to increase the color depth of the graphic using another graphics processing tool, or else scale the graphic using one of Push's scaling presets. Scaling always results in a TrueColor graphic, so Push's filters are always available immediately after a scaling operation.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Push complains "hotspot misses ID information" and won't save

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}You have probably clicked and dragged a very small hotspot somewhere on your bitmap and failed to add either a metafile or a hotspot attribute (jump, popup or macro) to this hotspot. You'll have to find the stray hotspot and either add an attribute or erase its bounding box.

Find this stray hotspot by using the **Tab** key to tab one by one through all hotspots on the graphic. Once you track down the hotspot, use the **Delete** key to erase it. Unless you have other stray hotspots, you will now be able to save your .SEG file.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

Scaling or filtering went "one step too far"

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}Push's filtering and scaling features are relatively quick and simple, and no **Undo** feature is available for undoing the last operation. You will need to reload the original image file and start over with your scaling and filtering operations by selecting **Reload** from the **File** menu.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;separat2.bmp}

SEG hotspots won't execute the desired action in the compiled project

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}This error should have been reported by HelpDebug when you loaded your help project (.HPJ) file into Push for analysis. One of several things could be happening. Here is a list of things to look for:

1. Have you checked your hotspot attribute information to insure that it is correct?
2. Have you updated HelpDebug's database since altering your project so that Push's drop-down list of context strings will reflect any changes you may have made to your source documents? You may be attempting to reference a topic whose ID has been changed or removed.
3. If the hotspot is a custom macro hotspot, are you sure the macro was registered correctly in your .HPJ file's [CONFIG] section?

4. Have you loaded and saved all .SEG files from older JPANI or LZANI projects? These .SEG files will not be compatible with ThinHelp and must be converted with a simple load-and-save operation.
5. Have you checked related links, macros or other types of jumps as a means of narrowing down the cause of the problem?
6. Are you trying to use full-length macro names in a 32 bit helpfile? 32 bit Help will not function properly unless standard Help macros in .SHG hotspots, .SEG data files and embedded window statements are referred to by their abbreviated names. **JumpID()**, for example, must be referred to as **JI()** in order to function. The compiler will automatically abbreviate macros coded in your RTF source, but it will not abbreviate long-form macros coded in SHG files, embedded window statements and .SEG files.

A .BMP file that looks fine in Push won't show up in the compiled Help or Viewer title

{ewl THNHLP,THIN,!PUSHBMPS.LZH;DOCBAG2.BMP;0/0/0/0/0/255/0}If you're sure that your embedded window syntax is correct and the embedded window reference is not used in a table, then you have probably inherited an RLE compressed bitmap and tried to display it as an embedded window. (If it *is* used in a table, you probably have a table formatting compatibility problem.)

RLE compressed bitmaps import correctly into Push, but they're not compatible with the ThinHelp DLLs. Load the bitmap into Push and save it, unaltered, as a .BMP using **Save Picture As...** to find out for sure. If the saved file is larger than the original .BMP, you more than likely tried to embed an RLE-compressed bitmap without realizing it. If there is no change in file size, try repacking the saved file into the LZH archive and recompiling anyway. There may have been a "bogus byte" somewhere in the file and the load-and-save operation could have repaired it.

Other problems

{ewl THNHLP,THIN,!PUSHBMPS.LZH;HLP-BUTT.BMP;0/0/2/0;;JI(`thinhelp.hlp>main'~`ID_TROUBLE')}Detailed help with other ThinHelp- and Push-related problems is available from the main troubleshooting menu linked to the help button.

If you have been unable to get a .SEG hotspot to function in your project even after checking all these possible errors, contact us about support. It is possible that you have discovered a bug in the software or an error in the user's guide which resulted in the problem.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;TOP.BMP;0/0/34/0;;JI(`'~`pushtrouble')}

{ewl THNHLP,THIN,!
PUSHBMPS.LZH;NOTE.BMP;0/0/0/0/0/255/0}
HTML authors take note: WinHelp supports rectangular hotspots only.   Polygons and freeform hotspots are not supported.   To mimic the effect of a polygon hotspot, use several adjacent rectangular hotspots arranged to cover the desired target area.

{ewl THNHLP,THIN,!
PUSHBMPS.LZH;MAGGLASS.BMP;0/0/0/0/0/255/0}Click **Zoom** from the top menu of Push and select the appropriate zoom level.   **1** is one-to-one scaling, **2** is two-to-one (twice as large as standard display), etc.   You'll find Push much faster to work with as a hypergraphics editor once you master the zoom hotkeys (**Alt+1** through **Alt+4**).

Zooming in on a bitmap does not mean your bitmap will be saved the way you see it when zoomed.   Your bitmap will only be saved in a different size if you select one of the three **Scale** options from the **Edit** menu.

{ewl THNHLP,THIN,!
PUSHBMPS.LZH;CAUTION.BMP;0/0/0/0/0/255/0}
Existing .SEG files from previous JPANI/LZANI projects are not compatible with ThinHelp.   They will need to be updated to conform to the new ThinHelp format.   Updating .SEGs is as easy as loading a SEG.EXE-generated .SEG file into Push and saving it...no other steps are required.

{ewl THNHLP,THIN,!
PUSHBMPS.LZH;NOTE.BMP;0/0/0/0/0/255/0}
When ThinHelp reduces your bitmap’s color depth to 236 colors, it is actually creating a graphic with only 236 *custom* colors.   The remaining 20 colors are Windows’ standard palette colors, reserved by the system.

Push’s registry information

{ewl THNHLP,THIN,!PUSHBMPS.LZH;REGICON.BMP;0/0/0/0/0/255/0}Push writes two values to the registry, one for the default HPJ file and one for the default window size as shown here. This diagram shows how to locate these entries using RegEdit if you should ever need to erase them or modify them manually.

{ewc THNHLP,THIN,!PUSHBMPS.LZH;REGISTRY.BMP}

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

The SEG file format

```
/* SEG.H -------------------------------------------------------*
 * This Defined the SEGDATA-Structure used by Push and JPANI     *
 * To Store Data-Correspondence between Position on the Screen*
 * and Help-Topic                                 *
 *------------------------------------------------------------*/

#ifndef __SEG_H
#define __SEG_H


/* File structure of .SEG files had to be revised between JPANI
Version 1.x and ThinHelp. The new .SEG-Files are not compatible with
the old ones and old files have to be converted by PUSH.EXE
   The main reason for the change in the file structure is that we
felt it would be necessary to support longer macro strings, especially
with respect to Windows Help V4.0. PUSH.EXE originally permitted only
128 characters in macro strings.
   There are also more dummy data fields for future enhancements as we
plan to implement several additional Features.
   To show the incompatibility of the Files the new SEG-Files to now
use a different SEG_MAGIC number.
*/

#define FILENAMELENGTH     256        // Maximum length of a filename
for .HPJ and .JPG/.BMP-File
#define MAX_SEGINFOS    600      // Maximum number of SEGJUMP entries
in one SEG file.
#define MACROLENGTH     8000          // Maximum length of a macro-
string... more or less a random number...
#define HOTSPOTIDLENGTH 200      // Maximum length of a topic context
string... more or less a random number...

#define SEG_MAGIC     12199           // Magic number to show it's
really one of our SEG files

#define SI_JUMP  0               // Marks hotspot type: Jump
#define SI_POPUP 1               //               Popup
#define SI_MACRO 2               //               Macro

/* SEGSTRINGs for macros and context strings have unlimited string
lengths.
   When loaded into memory, the application program (THNHLP.DLL)
stores a pointer to the string on the hard disk. An offset into the
SEG file is to be used instead.
   The end of the string is to be detected by the trailing zero as
usual in C.
   Note that the string's length is not limited by the way this file
format is defined.  THNHLP.DLL 16 bit will not allow the use of more
than about 4-8 Kbyte of strings at one time.
```

```
*/
typedef union
      {
          DWORD  dw;              // Byte offset from the beginning
of the SEG file
          LPSTR  lpsz;                    // Pointer to the string
when loaded into memory by PUSH.EXE
          HLOCAL hlocal;          // Local memory handle used by
THNHLP.DLL
        } SEGSTRING;

/* SEGJUMP structure defines one rectangular hotspot area that is
connected to a context string or macro reference.
*/
typedef struct
      {  /* Rectangular area for the segment.
              Note: This is NOT binary-compatible to a RECT structure
on WIN32 */
           short left, top, right, bottom;

           /* WinHelp macro string to be executed when the user clicks
on the hotspot.
              For SI_POPUP and SI_JUMP this member contains the
context string used for the jump */
         SEGSTRING    macro;

           /* The hotspot ID is the string that SHED also allows for
entry for our hotspot. It may be used later for special effects. */
           SEGSTRING  HotspotId;

           /* Metafiles can be huge pictures and are stored by
position in the file. */
           DWORD      metapos,      // Position of the metafile in the
SEG file
                      metalen;      // Length of the metafile
information in the SEG file.

           /* Style-Control Flags */
           DWORD      flags;                 // SI_xxxx Style Flags

           /* Variables reseved for future definition. Must be zero.
*/
           DWORD dwReserved[9];
      }
      SEGJUMP, FAR *LPSEGJUMP;

typedef struct
      {
           short Magic;                       // Should Contain
SEG_MAGIC
           short SegJumpCount;               // Number of Segments
currently available
```

```
           short Width, Height;            // Width and height need
to show ALL Hotspots

           //--------- Design-time-only information
----------------------------
           char  PictureFileName[FILENAMELENGTH];// Filename of the
picture file that shall be shown at design time
           char  HpjFileName[FILENAMELENGTH];    // Filename of
the .HPJ - File containing the Help topic jump IDs

           //--------- Help THNHLP to save memory
------------------------------
           DWORD HeaderAndStringSize;        // Size of
SEGINFOHEADER+SEGJUMP+strings without metafiles

           /* Variables reserved for future Definition. Must be Zero.
*/
           DWORD dwReserved[10];
      }
      SEGINFOHEADER, FAR *LPSEGINFOHEADER;

/* A SEG_File consists of one SEGINFOHEADER structure followed by
SegJumpCount SEGJUMP-Structures, which may be followed by metafile and
string data.

   There is no common metafile or string information between the
SEGJUMPs or the SEGINFOHEADER and the SEGJUMPs

         +-------------------+-----------------+
  0x000: | SEGINFOHEADER     | SegJumpCount=3  |
         +-------------------+-----------------+
  0x208: | SEGJUMP           | macro.dw=0x2aa  | --\
         +-------------------+-----------------+   |
  0x23e: | SEGJUMP           | macro.dw=0x2b2  | --+--\
         +-------------------+-----------------+   |  |
  0x27e: | SEGJUMP           | macro.dw=0x2bd  | --+--+--\
         +-------------------+-----------------+   |  |  |
                                                   |  |  |
  0x2aa: "Print()\0" <-----------------------------/  |  |
                                                      |  |
  0x2b2: "JI(`',`ID_CONTENTS')\0" <-------------------/  |
                                                         |
  0x2bd: "About()\0" <-----------------------------------/

   PUSH.EXE sorts the SEGJUMP records by the size of the rectangular
area, so large areas will be at the end of the file. Thus scanning
from the beginning of the file will find the smallest SEGJUMP matching
a given point.
   The maximum length of the file excluding metafiles is 65535 Bytes,
thus limiting SegInfoCount to MAX_SEGINFOS.
   The metapos and metalen fields of a SEGJUMP give the starting-
```

```
Position of a Metafile byte relative to the beginning of the File. The
metafile data is metalen bytes long.
   The "PictureFileName" and "HpjFileName"-Members are only used
during SEG editing. They are not required by ThinHelp.
*/
#endif
```

{ewc THNHLP,THIN,!PUSHBMPS.LZH;PREVMENU.BMP;0/0/34/0;;JI(`'~`IDH_PUSHHELP')}

About to launch Help to RTF...

All files in ThinHelp's help system include complete, uncrippled functionality with our Help to RTF conversion software for allowing you to create Word-compatible documentation from helpfiles including all bitmaps and embedded window objects.   This feature is reserved exclusively for helpfiles included with Herd Software Development products.   For complete functionality with all helpfiles you will need to own a license to Help to RTF Standard or Help to RTF Pro.

You can use the evaluation version of Help to RTF included in this package to create printable documentation from any of the included helpfiles and optionally load the appropriate helpfile directly into Help to RTF by selecting the **Print Some or All Topics...** item from each helpfile's **File** menu.

Select one of the following:

Launch Help to RTF   **Learn more about Help to RTF**