PEG®

Portable Embedded GUI

Development Toolkit User's Manual

Second Printing

October 2007

© Copyright 2005-2007
Swell Software, Inc. All rights reserved.

© Copyright 2005-2007

Swell Software, Inc.
3081 Commerce Dr.
Fort Gratiot, MI 48059
PH: (810) 385-2893
FAX: (810) 385-2947

info@swellsoftware.com

No part of the document may be reproduced in any form without the express written consent of Swell Software, Inc.

All rights reserved.

PEG® is a registered trademark of Swell Software, Inc.
C/PEG & PEG+™ are trademarks of Swell Software, Inc.

TABLE OF CONTENTS

Forward .... **iii**
Introduction .... **v**
What PEG IS vi
What PEG is NOT vii
Where PEG is going viii
Library Updates ix
Chapter 1
PEG FontCapture **1**
Configuring Character Range 5
Multilingual Support and UNICODE 6
What is UNICODE? 6
PEG Character Encoding 7
Should You Use UNICODE? 9
Defining Unicode Strings 10
Using Custom Fonts 11
Chapter 2
PEG ImageConvert **13**
Overview 13
Input File 15
Output File 16
Compression 16
Palette Options 17
Output Format 20
Screen Rotation 22
Transparency 23
Output Colors 24
Batch Conversion 25
Implementation Notes 26
Chapter 3
PEG WindowBuilder™ **27**
Overview 27
PEG WindowBuilder Project Files 27
Source Output Files 28
Screen Layout 29
The Project Window 30
Project Window Menu Commands 31
Working with Modules- The Source Page 37
Working with Images- The Images Page 40
Working with Fonts- The Fonts Page 41
The Target Window 45

Selecting Objects in the Target Window 46
Target Window Menu Commands 46
The String Table Editor 52
Merging String Tables 57
Exporting the String Table 57
Source Code Generation 58
Pointer Name Control 59
Example 1: Creating a simple PegDialog window 61
Creating and Configuring a Project: 62
Editing the Module: 68
Saving Your Work: 71
Examining the Source Code: 72
Example 2: An advanced PegDialog window 72
Customizing PEG WindowBuilder 80
Step-by-Step 82
The SwellButton and SwellScale 86
Building wbuser.dll 88
Appendix
PEG Directory Structure 89

FORWARD

We at Swell Software thank you for choosing PEG+!

The authors of PEG are first and foremost embedded systems programmers like yourself. With extensive experience developing software for closed-loop servo robotics, industrial control systems, measurement and monitoring equipment, and consumer electronics, we most likely share many common experiences with you. We believe this kinship will allow us to anticipate your requirements and to provide you with the tools and support you need as you develop your next product.

In addition to the PEG development package, Swell Software provides consulting and contract programming services to clients in a wide diversity of industries. These services range from one-day on-site evaluations and tutorials to complete screen prototyping and development. We encourage you to take advantage of these services as early as possible in your project cycle. If you have purchased or are evaluating the PEG library, you can of course contact us at any time via phone or email to answer your technical questions.

PEG is currently being used in projects around the globe, yet PEG also continues to grow and improve. Minor updates are often published every few weeks as we incorporate suggestions and requests from PEG users. We encourage you to provide us with feedback as you begin using PEG in your application development. If you find something just isn't working for you, or we are missing something you feel is a requirement, please do not hesitate to let us know. We will make every effort to satisfy your request and provide you with an updated release in as short a time frame as possible. We are committed to making your embedded development effort an overwhelming success!

How are the manuals organized

The PEG+ Programming Manual is organized such that the manual explains the configuration and build procedures you will need to know in order to begin using the PEG library. This allows you to be up and running and experimenting with the library very quickly.

The remainder of the Programming Manual provides an 'under the hood' view of PEG library internals and introduces basic concepts that are needed to fully understand how PEG works. You will need to read and

understand this material before you begin serious development of your application level software. This is followed by descriptions of the fundamental PEG classes. These descriptions contain many working examples that will prove valuable to you as you begin writing your own system software.

The second manual, the PEG Development Toolkit User's Manual, describes our supporting cast of utility programs. These include PEG FontCapture, PEG ImageConvert, and PEG WindowBuilder. The appendices describe the PEG installation directories and the example programs.

The third manual, PEG+ API Reference Manual provides extensive information about the fundamental PEG classes. This manual details the Application Programming Interface (API) of the PEG+ graphics library. It is intended as a quick reference guide for developers which may already be familiar with how PEG+ works and may need to review details on individual functions. The Reference Manual is also provided in interactive PDF format. We believe the PDF format class reference is more convenient to use on a day to day basis than the printed manual. This approach also works well in that as you read this manual you are not overloaded (no pun intended!) with member function names and descriptions. Instead, we encourage you to first concentrate on obtaining a high-level understanding of how PEG works. Later, as you begin working on your system software, you will probably want to keep the API Reference Manual open at all times.

The Programming Manual and PEG Development Toolkit User's Manual are also provided in PDF format, and the PDF format often contains last minute changes or additions that you will not find here. These additions will be noted in the Release Notes. The online manuals are found at the following address:

http://www.swellsoftware.com/download/documentation.php

Username and password are required to download the manuals.

INTRODUCTION

Historically speaking, graphical user interfaces have almost exclusively been the domain of desktop personal computers. This has been the result of two main factors: the cost of graphical display hardware and the lack of GUI software suitable for use in real-time systems.

In the area of industrial control systems, there have been attempts at providing graphical presentations, but these have been cumbersome at best and terribly expensive as well. These types of systems have typically avoided the use of mainstream video output devices, and opted instead for very expensive and functionally limited industrial display terminals.

Today, this attitude has changed to the point where it is very common for an embedded system to contain many of the very same hardware components found in a desktop computer system. This makes sense strategically because it allows the inventor of an embedded product to leverage the sales volume and pricing of the components sold primarily for desktop computer use. The result is that the cost of including graphical display hardware in an embedded product has declined significantly over the past few years. A wide variety of LCD display panels, VGA display panels, video controller chips and high-performance CPUs capable of driving a graphical interface are now available.

Unfortunately, the software side of the equation has not advanced nearly as quickly. Until now there has been no graphical interface solution that is small enough and portable enough for an embedded system while at the same time providing a modern and professional appearance. There have been previous attempts at meeting this need, but so far these attempts have missed the mark.

The alternative solutions that provide a modern, full-featured interface have all been derived from desktop computing environments, and carry along with them years of acquired baggage. These solutions impose very high hardware costs on your system, and even higher costs in terms of the man-hours required to successfully integrate these large software packages with your real-time software. This of course assumes you have the time and expertise required to actually build a working system with one of these products. We have seen more than one project descend into a never-ending abyss of delays, technical setbacks, and finally failure caused by trying to force-fit software that was not intended for real-time systems.

We believe you deserve a better solution!

What PEG IS

PEG is an acronym for Portable Embedded GUI. We chose this name because we believe it accurately reflects the design and motivation that went into the creation of our development package.

PEG is Portable

We have designed our software to be portable to any target hardware that is capable of graphical output. PEG does not expect or require any underlying software components in order to do its job. If you have a C++ compiler and hardware capable of pixel-addressed graphical output, you can run PEG.

PEG is Embedded

This statement is rather vague, because it means so many different things to different people. The bottom line is that ***PEG is, and will always be, targeted only at real-time embedded systems.*** This distinction is so important that we felt it should be included in the name of our library.

PEG is GUI

The PEG class library provides the building blocks for a powerful and extensible graphical user interface. Users of PEG will find that they can create a graphical presentation rivaling anything on the market today. Extensive thought and research have gone into the design of our product to insure that you are receiving a library that is fully capable of supporting all of the advanced GUI features you need today, while also accommodating future enhancements. Advanced clipping techniques, font support, graphic image support, and smart object methodologies are incorporated in our library. We are confident that the internal design of the library is such that PEG can grow and advance for years to come while building on the existing foundation.

In addition to the class library itself, PEG provides all of the other tools, documentation, and support you will need to construct a custom graphical interface for your project. This includes utilities for generating graphical fonts ***(PegFontCapture)***; processing, optimizing, and compressing graphical images ***(PegImageConvert)***; and ***PegWindowBuilder***, a RAD prototyping tool for use with PEG. With the class library and related tools,

PEG without question provides the most powerful, professional, and complete GUI solution available to real-time embedded system developers.

What PEG is NOT

The large software companies are today providing software that is intended to be a 'one size fits all' solution. This has led to some confusion among many developers concerning what a GUI library should do, what it should not do, and what components are required to build a working system. We believe it is worthwhile addressing these questions up front to insure that we are all working from the same starting point.

PEG is not an operating system. PEG provides no code for task switching, memory management, resource management, or inter-task communication. Contrary to popular belief, the desktop windowing environments are not part of the operating system either. This should be obvious from the fact the graphical environment can be significantly updated without improving or otherwise changing the underlying OS. In order to build a real-time multitasking system using PEG, you will also have to incorporate an operating system kernel. PEG is already fully integrated with most of the leading real-time operating systems available today. PEG can easily be integrated with other operating systems as well, and PEG can run standalone if multitasking is not required for your application.

PEG is not an application program. The PEG library, by itself, will provide an end user with absolutely zero in terms of useful interaction or information display. It is your job to create the windows, dialogs, and other objects that will be used to retrieve input from and display information to the end user. Of course, the whole point to using PEG is that our library provides the tools and components that make creating your application level interface a manageable task.

Finally, PEG is not a PC-library. While PEG does support common PC development environments as a matter of convenience and productivity, the goal of PEG is to provide a full graphical interface solution to real-time embedded systems developers. This solution includes the software, utilities, documentation and support required to make your embedded development effort a success, regardless of the final hardware implementation.

Where PEG is going

Over the near term, the core PEG library will continue to grow and improve in terms of the native control types that are available and the flexibility of those controls. Over the longer term, we plan to add entirely new groups of object types we feel would be useful for embedded systems. Of course, the long-range development list also depends on input we receive from you, the developer using PEG. In any event, the basic library will retain the ability of allowing you to remove unwanted components in order to build a system that exactly fits your size and performance requirements.

While PEG can be ported to nearly any hardware configuration capable of graphical output, this effort can seem confusing to a new user. For this reason, we have undertaken to add reference platforms for many common hardware configurations. This allows PEG users to begin running PEG immediately on hardware which is similar if not identical to the final target. Currently reference platforms have been completed or are in development for x86, ARM7, ARM9, StrongARM, MPC823, ColdFire, DragonBall, C167, and MC68332 platforms. Additional reference platforms will be added as evaluation hardware platforms become available for other CPU types that are popular in the embedded community.

PEG WindowBuilder, our newest and most powerful support utility, is continually being enhanced to meet the needs of PEG users. Fully integrated support for Unicode and efficient code generation through the use of container classes are new features as of this manual printing.

Library Updates

Library updates are posted on the Swell Software www site roughly every 90 days. If you are a new PEG customer, you are entitled to a minimum of three months of technical support and library updates. The Download/ Updates page on the Swell Software Website is password protected. If you do not know the password, please email support@swellsoftware.com and request the current password.

The http://www.swellsoftware.com/download/updates.php page lists the most recent changes or library enhancements, and also allows you to download the latest release of PEG library source code, supporting utilities, and documentation. This website has gone through an enhancement phase and now includes a Frequently Asked Questions (FAQ) page with useful contributions from current PEG users.

CHAPTER 1

PEG FONTCAPTURE

PEG FontCapture is a utility program written using PEG that can be used to generate additional fonts for use by your application program. PEG FontCapture is included with licensed distributions of the PEG library. PEG FontCapture generates .cpp source files or binary data files each containing a PegFont data structure, character widths and character data. Versions of PEG FontCapture are available for MS Windows (all versions) and most Unix/Linux/X11 development stations.

When you choose the standard source-file output format, the .cpp files generated by PEG FontCapture can be compiled, linked, and if desired ROM'ed along with the rest of your application code. Since the .cpp files generated by PEG FontCapture entirely contain read-only data, additional PegFonts included with your system software should only consume ROM storage.

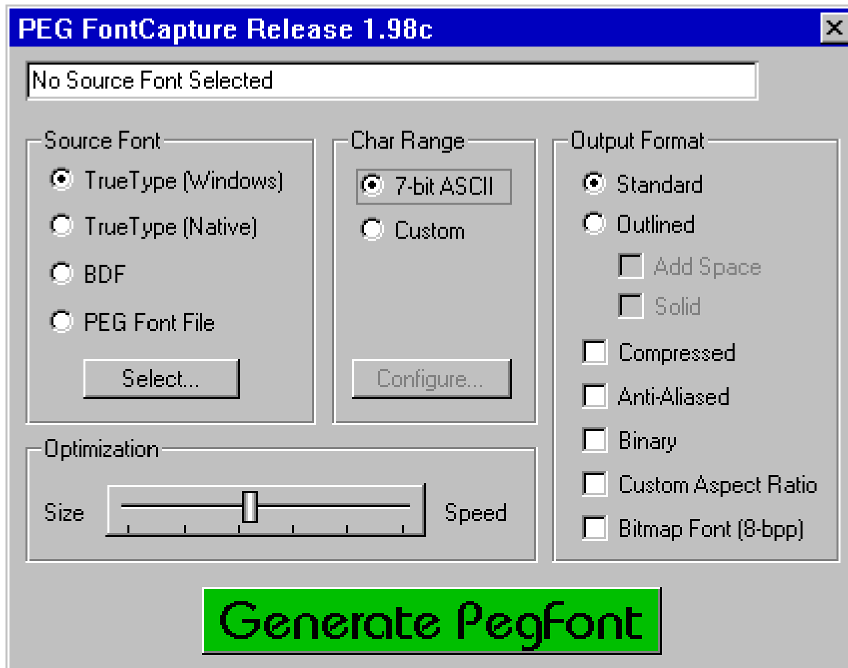
The binary output format is intended for use with targets which support a run-time file system. In this type of system, an arbitrary number of PegFonts may be stored as binary files and read from the file system into memory only when needed.

The PegFont format is a binary 1 bit, 2 bit, or 4 bit per pixel font format. The standard 1-bpp format is most commonly used because it requires the smallest memory size. These fonts can be drawn in any combination of foreground and background colors supported by your target system.

The 2 bit/pixel format is used for Outline fonts, which is a font format used primarily in video display systems such as television or set-top box applications. In this format the font includes a single-pixel wide outline. When drawing the font, the foreground color is used for the font outline and the background color is used as the font fill color.

The 4 bit/pixel format is used for anti-aliased fonts. Anti-aliased fonts define 16 pixel intensity levels to produce much smoother appearance for round character edges.

PEG FontCapture appears as shown below:



The ***Source Font*** group allows you to select the source font type. FontCapture supports the conversion of MS Windows TrueType fonts or Adobe Postscript **Glyph Bitmap Distribution (BDF)** fonts.

If you are converting a TrueType format source font, you can use either the native MS Windows font rendering engine or the open-source FreeType (Native) rendering engine. These two font renderers produce slightly different results due to internal differences in implementation.

If you select TrueType (Windows) as your source font type, the Select… button can be used to invoke the ChooseFont Windows common dialog (Note: this option is only available when running FontCapture on a MS Windows host).

If you select the TrueType (Native) as your source font type, the Select... button is used to choose a .ttf file from your host system file system. Note the Native rendering engine works directly from a TrueType source file, it does not utilize the “Installed Fonts” of your Windows host system. This rendering engine can be used both on Windows, Linux, and Solaris host environments.

If you select the BDF source font format, the Select button allows you to select the actual .bdf file from your development station file system. No matter which type of source font you choose, the output of Font Capture is a PegFont data structure in ‘C’ data array format or binary format.

The final source font format is a PegFont file. This selection allows you to read a previously converted PegFont source file. If you have edited or customized the output of a PegFont and saved the font in source file format, this option allows you to re-open the source file for further editing and modification.

The ***Char Range*** group allows you to specify precisely the characters you want to include in your PegFont. For example the desired range for a certain font may include only numeric characters to reduce the resulting font size. For multilingual applications, you may need to specify a complex set of character ranges to support all languages included in your system. This sometimes involves using Unicode character encoding, which will be described in detail in later paragraphs. The simplest Char Range to specify is the ASCII character range. This includes characters 0x00 to 0x7f. For any other character range, you must specify in detail the range or ranges of character you want to include using the range configuration dialog described below.

The ***Output Format*** group allows you to select what type of conversion output you desire. The ***Standard*** format is a 1-bpp font saved in C source code format.

The ***Outline*** checkbox can be used to generate a font with an added single pixel wide outline of each glyph. This is NOT the typical font type used in PEG applications, but is supported for the minority of applications that require an outlined font capability. When the Outline box is selected, PegFontCapture encodes the output PegBitmap in a 2-bpp format; where bitmap value 0 indicates the pixel should be the foreground color, bitmap value 1 indicates the pixel should be in the outline color, and bitmap value 2 indicates the pixel should be either the background color or transparent, depending on the PegColor.uFill value passed to the text drawing function.

While FontCapture can generate 2-bpp fonts, you should not attempt to use them unless your PegScreen interface class supports this font format. The Win32 screen interface class PegWinScreen includes this functionality as a reference for users who desire to display outlined fonts.

The ***Compressed*** checkbox indicates that the PegFont will be created with a compressed data section so that it takes up less space in ROM. The PegZip function is used for the compression. In order to use a compressed font in your application, it must first be decompressed. To do this, you must first copy it into a new PegFont structure in RAM. Then use PegUnzip to decompress the data section into your new PegFont. So, while compressing your fonts will take up less ROM space, it will also take up a considerable amount of extra RAM space. Note that C/PEG does not support the PegZip/PegUnzip functions, so it therefore also does not support compressed fonts.

The ***Anti-Aliased*** checkbox can be used to generate a font that uses 16 intensity levels to produce a much smoother appearance for round character edges. The output PegBitmap is encoded in a 4-bpp format. You should make sure that your PegScreen interface class supports this font format, as the actual implementation of the anti-aliased drawing is dependent on your screen driver. The Win32 screen interface class PegWinScreen includes this functionality as a reference for users who desire to display anti-aliased fonts. In our example implementations, if your screen's color depth is 8-bpp palette mode, then the 16 colors used to draw the characters are the last 16 colors of the palette. Or if you are using a 16 or 24-bpp screen, then a calculated blend between the foreground and background colors are used.

The ***Bitmap Font*** checkbox is used to generate a font that is stored as an 8-bpp bitmap. The advantage to this approach is that PEG only needs to call the fast-drawing BitmapView function for each character, instead of parsing out a stream of bits. The disadvantages are that obviously the font will be 8 times the size of a standard 1-bpp PegFont. Note that your screen driver needs to have an appropriate function to display this style of font.

The ***Binary*** output format is used when you want to create a binary PegFont file, rather than the more common C source file. If your target includes a file system, the Binary output format allows you to save your PegFont to a file, and read this font from the file system at runtime. An example program named ***BinFont*** is available to demonstrate how to load PegFonts from a filesystem at program runtime.

The ***Solid*** and ***Add Space*** checkboxes are modifiers for the outline font generation mode. The Solid checkbox causes the font outline to appear somewhat heavier than the default outline. The Solid choice is beneficial when working with large fonts. The Add Space option adds a single pixel of spacing between each generated character when generating an outline

font. This is beneficial when working with very small outlined fonts. The Solid and Add Space modifiers are ignored if the Outline checkbox is not selected.

The ***Custom Aspect Ratio*** checkbox can be used to adjust the dimensions of the characters in your font. When the font is generated, a dialog will open that displays the current dimensions of the font in a grid form. Vertical and horizontal sliders are used to adjust the shape of the grid. Increasing the width or height of the grid makes the characters appear wider or taller in the font.

The ***Optimization*** slider gives you control over how the FontCapture program breaks mutiple-page fonts into component pages. The FontCapture program examines the resulting font and attempts to find sections of the font which contain no input data. When large empty sections are found, FontCapture removes the empty section by creating two font pages that skip the empty font section. The optimization slider allows you to adjust how large the empty section needs to be to justify creating a new font page. Creating more pages reduces the font storage size, but also increases the runtime overhead when displaying string because the correct font page for each character must be located at runtime. Note optimization works best when using the TrueType (Native) rendering engine, as more precise font data is available when using the Native rendering engine.

The ***Generate PegFont*** button causes FontCapture to convert the source font into a PegFont. This process may occur very quickly for a small font, or it may take several minutes for a very large font containing many thousands of characters. You can capture as many fonts as you like within one session of running FontCapture.

After converting the source font, PEG FontCapture will display the window below to preview the resulting PegFont:

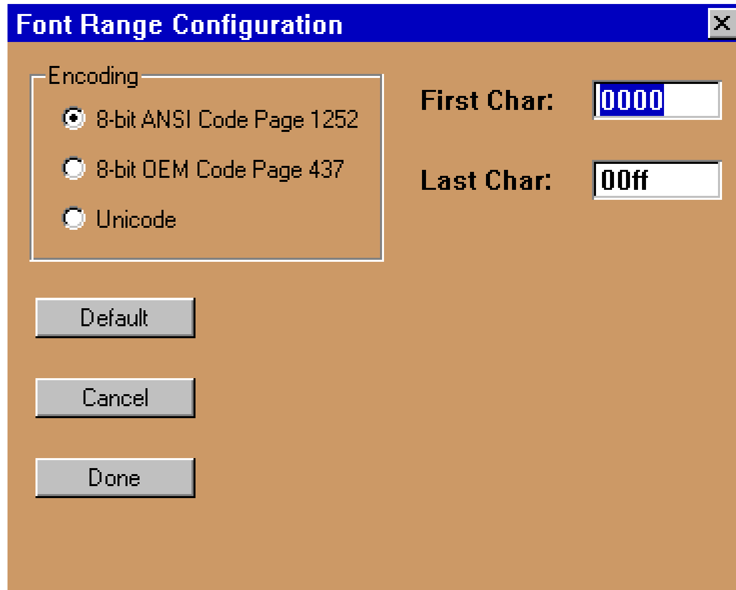
You can use this window to examine the PegFont produced, and even compare multiple fonts to find the best appearance. There are also 2 more ways that you can modify the font, even though it's already been generated. First, by pressing the ***Edit Font...*** button you can adjust the extra padding that is included on the top and bottom of all of the characters in the font. And second, pressing the ***Edit Character...*** button opens a dialog that allows you to actually modify individual characters right down to the pixel level. Once you are satisfied with the appearance of your font, you can use the ***Save As...*** button on this window to save the font to a source or binary file of your choosing.

1.0.1 Configuring Character Range

The ***Char Range*** group allows you to specify the range of character glyphs that will be encoded in the output font. When the ASCII option is selected, the range of characters is fixed to ASCII-0 through ASCII-127, which is the normal range for single language applications.

PEG Font Capture also allows you to specify a custom range of characters to be encoded. When you select the Custom option, the Configure… option becomes active, allowing you to fully define the range of glyphs which will be recorded in the output file.

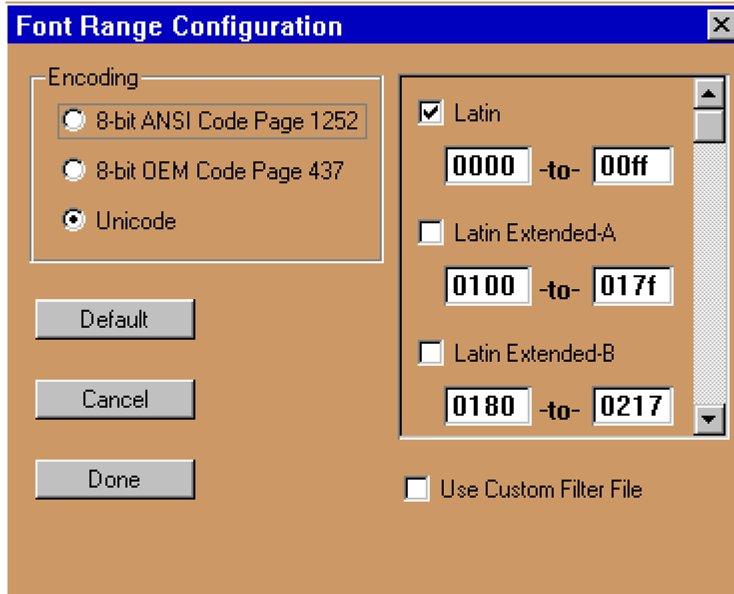
It is often the case a particular font is only used to display a certain range of characters; for example you may define one font that will be used only for displaying numbers. In this case, you do not need or want to encode the entire ASCII character range in the output file. Instead, you can enter a limited character range by selecting the “Custom” button, and entering the range of characters in the Range Configuration dialog, shown here:



The First Char and Last Char fields allow you to define the start and ending characters to be encoded. Using the numerical example above, you could enter “0030” (i.e. ASCII-‘0’) as the first character, and “0039” (i.e. ASCII-‘9’) as the last character. This will save quite a large amount of memory over capturing the entire ASCII character set.

1.0.2 Multilingual Support and UNICODE

A more advanced use of the Range Configuration dialog deals with UNICODE fonts. When you select the UNICODE option on this dialog, the dialog appearance changes as shown below:



Before we can fully understand how to configure custom UNICODE character ranges, we must first examine what UNICODE is, the options available for supporting multiple languages, and the trade-offs involved with each approach.

1.1 What is UNICODE?

If you are a software developer from North America, you may not be more than vaguely familiar with what UNICODE is, or what it means to your software. UNICODE is a standard definition of 16-bit character encoding that encompass all characters used for all of the most prominent writing structures. For example, the UNICODE standard defines character encodings for characters used to record Latin (~English), Japanese, Korean, and Georgian writings.

To really understand what the UNICODE is, a little clarification in terminology is required. We often confuse or mix the terms 'language', 'alphabet', 'character', and 'glyph'. A glyph is a shape representing a character. For example, 'A', '***A***', and '**A**' are three individual and unique glyphs, however they are all the same character: 'Capital Letter A'.

UNICODE defines a unique encoding for each character. UNICODE does not define a font, style, size, or any other attributes for a character. Since there are far more recognized characters in the world (> 28,000) than can be encoded using an 8-bit representation, UNICODE uses 16-bit values to encode each character.

Further Reading

To learn more about the UNICODE standard, we encourage you to purchase The Unicode Standard, Version 3.0 (ISBN 0-201-48345-9)

1.1.1 PEG Character Encoding

The UNICODE font range selection dialog allows you to specify the groups, or code pages, of characters you want to encode. If you select multiple code pages for one font, PEG FontCapture will generate at least one PegFont page for each code page you enable. In all cases the resulting fonts use Unicode character encoding, even if your code page selections leave “holes”, i.e. even if you select a non-contiguous set of character pages. However, the multi-page PegFont encoding scheme allows the final font to simply skip any unused range(s) of characters, eliminating memory use for those unsupported code pages.

Font Range Configuration

As stated above, PEG FontCapture allows you to specify precisely the code pages and ranges of characters you need for your application. You enable or disable each code page by selecting the corresponding check box for each page. The numeric range for a code page that is not enabled is ignored.

For each code page that is enabled, you can specify an exact window of character values to capture. These character ranges are entered in hexadecimal format, consistent with Unicode encoding.

The ability to capture limited windows within each code page is very useful for multilingual applications that are attempting to produce a minimal memory footprint. This enables you to select the specific code pages and ranges of characters required in your application, without capturing all of the characters in each page. For example, you may desire to capture code page 1 (Basic Latin) indexes 0020 through 0080, code page 2 (Latin 1) characters 0090 through 0100, and a few additional characters from code page 9 (Cyrillic). You may thus create a custom font containing <= 256 characters, but still containing all of the glyphs you need for your multilingual application.

Even if you are using 16-bit character encoding, you will very likely not want to attempt to capture the entire UNICODE character set. Such a character set would require a huge amount of memory, and it is highly unlikely that you will find a font containing anywhere near the entire UNICODE character set. PEG FontCapture allows you to specify exactly which code pages you want to capture from the selected font.

Once you have entered the range configuration, PEG FontCapture saves the configuration (or 'profile') to a binary file for later retrieval. The next time you start the PEG FontCapture program, it will automatically default to the set of ranges defined in the previous usage.

Whenever PEG FontCapture is operated using a Custom font range, the header in the output file produced contains a comment section indicating the mapping of Unicode characters to the character indexes in the captured font. This mapping is required if you want to manually enter 16-bit string values in your application.

Applying Custom Character Filters

FontCapture also allows you to specify a custom range of characters to be encoded by using a character filter file. On the Range dialog you may select the **Use Custom Filter File** checkbox. When this checkbox is selected, you can type the path and filename of a file to be used as a final character filter. In other words, characters selected above will be verified against the filter list and only those characters listed in the filter will be included in the output font.

The custom filter file should have one hexadecimal character encoding per line. An example is shown here:

```
0x3456         // you can put comments
0x3467         // or other notes after the character encoding
0x3786         // as shown here
```

This file format was chosen because it works perfectly with the encoding tables provided with the Unicode Standard version 2.0!! The Unicode standard accepts that many character encoding "standards" are in existence and provides tables to map the alternate character encodings to the Unicode encoding. These tables can be directly supplied to PEG FontCapture as filter files, allowing you to generate Unicode encoded fonts containing only those characters defined by a previous standard.

1.1.2 Should You Use UNICODE?

If you are working on an application that must support many languages, this is of course the question you are anxious to answer. Supporting multiple languages does not always imply using multiple alphabets or using a single character set containing all of the characters required for each language. All common North-American and most Western European languages can be supported very well by using a single 256 character alphabet. There are, of course exceptions.

There are two schemes in broad use for displaying information in multiple languages. The first scheme uses multiple 256 character font sets, with the characters required for each language or possibly each group of languages encoded in separate font files.

For example, suppose you write down and tabulate every character required to display all strings defined in your application in each language you are required to support. After going through this process, you may find your application needs to be able to display a total of 500 unique characters. Further, you find half of the languages can be supported using the first 250 characters, and the other languages can be supported using the second set of 250 characters. In this case, you might decide to use two 256 character font sets, and switch languages simply by switching the fonts used for displaying strings in different languages.

The advantage to the multiple-font approach is you do not need to use 16-bit character encoding, although your application can draw more than 256 unique glyphs. This can simplify your application development and reduce the amount of memory required to store your character strings. One disadvantage of this approach is that there is not a unique mapping of characters to character encoding, i.e. character 0x0030 in one font may be the glyph ‘0’, while in another font this may be an entirely different glyph.

The alternate approach is to place all required glyphs in a single font file (or multiple font files, if different font sizes and styles are needed). If the number of characters contained in ANY single font file exceeds 256, you will need to run PEG in UNICODE mode, meaning all PEG strings will be encoded using 16-bits/character. Note we prefer here to use the term “16-bit encoding”, since as stated in the previous section the font produces with Font Capture may not contain a true Unicode character mapping, depending on the font range configuration. Running PEG in “Unicode Mode” means only you are using 16-bit character encoding, and does not mean your string values will be strict Unicode standard encoding.

The advantages to the UNICODE approach are you do not need to switch fonts when switching between languages, each character encoding is unique and unambiguous, and very large character sets are accommodated with no extra programming effort beyond what is required when first stepping to UNICODE. The disadvantages include increased memory requirements for string storage, and the inability of software debugging applications to display arrays of 16-bit encoded values as "strings." A further disadvantage is many run-time string libraries do not support 16-bit character encoding.

You should not take the decision to use 16-bit character encoding lightly. While this string encoding can greatly simplify the long-term programming effort, it will almost certainly take you and your development team a while to become comfortable with using 16-bit characters. Further, it is a time consuming process to manually enter strings that use 16-bit encoding, especially if your compiler has no intrinsic support for 16-bit character encoding. While the PEG WindowBuilder string table editor is provided to aid this process, you will miss the simplicity of entering "String" into your editor!

1.1.3 Defining Unicode Strings

If you decide to run in Unicode mode, you will need to define and initialize your string data such that it is recognized as an array of 16-bit variables. A few compilers have built-in support for this type of data entry, but even in these cases you cannot enter Unicode-formatted strings that use characters above the printable character range using the standard 'C' double quoted string syntax.

The easiest way to create your Unicode strings is to allow PEG WindowBuilder (described in chapter 12) to generate them for you. PEG WindowBuilder provides a drag-and-drop string entry editor that allows you to select any character from an extended font file you have generated using PEG FontCapture. You simply pick the characters you want to use, and WindowBuilder generates the associates Unicode formatted initialized array in a portable 'C' format.

If you really need to enter your own Unicode format strings, the most portable method is to enter the strings as PEGCHAR arrays, as shown below:

```
PEGCHAR TestString[] = {'H','e','l','l','o',' ',0x209,0x210,0x224,0};
```

In this example, we have mixed a few characters from the ASCII range with a few characters that are only available in the extended Unicode character set. Note there currently is no other method available for entering and initializing character codes which fall above the ASCII range unless you are using some type of Unicode enabled editor.

1.1.4 Using Custom Fonts

Using fonts generated with PEG FontCapture is very simple. Every PEG object which supports text output has a member function called SetFont(PegFont *). Therefore, after constructing a PEG object that should use the new font, you simply call that objects SetFont function, passing a pointer to your new font. For example, assume you told PegFontCapture to generate a new font called “MyNewFont”, by typing this in the font name field of PEG FontCapture. In this case, the following code fragment illustrates the process of altering the font used by a PEG object:

```
extern PegFont MyNewFont;

PegTextButton *pButton = new PegTextButton(10, 10, 100,
        MESG1, “NewFont”);
pButton->SetFont(&MyNewFont);
```

Likewise, if you have overloaded a Draw() function for a window or other object, and you are drawing text on the screen, you can simply pass the pointer to your new font to any of the ***PegScreen*** text information or output functions.

Changing Font Defaults

Every PEG object type that supports text has a default font definition. This default font definition is held in a static array which is a member of the PegTextThing class. By calling the PegTextThing::SetDefaultFont function you can change the default font assigned to all successive instances of any or all PegTextThing derived objects. For example, you might use the SetDefaultFont function to change the font used for all PegTitle or all PegTextButton objects.

CHAPTER 2

PEG IMAGECONVERT

2.1 Overview

PEG ImageConvert is a utility program developed by Swell Software that can be used to convert MS Windows .bmp, CompuServe GIF, PNG, and JPG files into binary or source code formats supported by PEG. All of these file formats are in the public domain, and you may use PEG ImageConvert to process files which you create in any of these formats. PEG ImageConvert is a program written using PEG and running under the Win32 or Linux development environment.

Input files can be 1, 2, 4, 8, 16, or 24 bit-per-pixel formats. Likewise, PEG ImageConvert can generate 1, 2, 4, 8, 16, or 24 bit-per-pixel compressed image files (Note that PEG ImageConvert for C/PEG does support 24-bpp input, but does not support 24-bpp output.) For output of 8 bit-per-pixel or less, the palette used to encode the output image can be saved in source format during the image conversion process,suitable for use in calling the PegScreen **SetPalette()** function.

The actual operation of PEG ImageConvert is heavily dependent on the selected output format. PEG ImageConvert may perform color reduction, dithering, RLE encoding, and transparency encoding for all input file types. In addition, for 8-bpp output format, PEG ImageConvert adds optimal palette generation.

While PEG ImageConvert can output PegBitmap structures in many formats, this is of little use if the PegScreen driver being used on the target is not capable of properly displaying the bitmaps in the format in which they are saved. For example, while PEG ImageConvert can save PegBitmap structures using 2-bpp encoding, this format should only be used if your derived PegScreen driver class understands and can properly display images saved in 2-bpp format. ***The PegScreen derived interface classes provided with PEG support both 8-bpp bitmap encoding and the native encoding corresponding to the color depth of the target display.***

All of the options that can be selected on the PEG ImageConvert dialog window control the **output** of PEG ImageConvert. The format and data content of the input file(s) is determined by PegImageConvert by reading and parsing the input file header information.

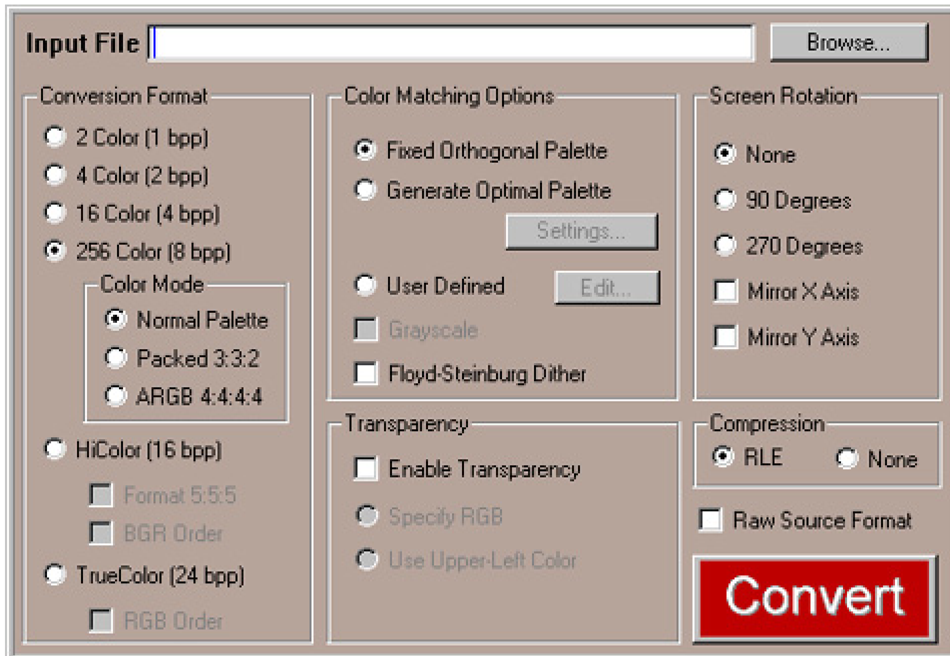
If your target supports 256 or more colors, PEG ImageConvert can also perform advanced palette reduction and optimization, allowing you to create and use any number of color palettes, each of which is optimized for the images displayed in your application. This option is best utilized along with batch image processing (described below), which allows a custom palette to be created for optimal display of multiple images. The input files for list processing can be any combination of the supported file types, and can even have different internal formats in terms of the color resolution associated with each input file. For those readers who are familiar with color quantization methods, PEG ImageConvert utilizes an improved form of Heckbert's 'Median Cut' algorithm for color reduction.

PEG ImageConvert is not a paint program or ray-tracing package. If you will be creating and using bitmaps and animations in your PEG application program, you will need to obtain a paint and/or graphics program capable of outputting one or all of the supported input file formats. The **Paint** program which is standard with MS Windows provides the minimum functionality you will most likely need, although **Paint** is very limited in terms of palette control and export formats.

If you are working with 256 or more colors, you many also find it useful to purchase an image processing software package capable of translating between several common graphic file formats. ***Paint Shop Pro***, by Corel, is one such graphics program that will do everything you will need when creating bitmaps for use with PEG. For this reason, a fully licensed version of ***Paint Shop Pro*** is included in your purchase of either PEG+ or C/PEG.

The Conversion Dialog

The PEG ImageConvert application dialog appears as shown below:



2.1.1 Input File

The input file string allows you to select the source image file. You can either type in the name of the file, or you can use the ‘Browse’ button to select a file from your computer.

Only one file path\name can be entered in the Input File string field. If you want to process multiple input images at one time, you should enter the image names in an ASCII command file and select the command file as the input file. This is described in more detail in the ‘Batch Processing’ section.

The basic input file type is determined by PEG ImageConvert based on the input filename extension. For MS or OS2 Bitmap files, the filename extension should be ‘.bmp’. For GIF files, the filename extension should be ‘.gif’, for PNG file the extension should be “.png”, and for JPEG files the filename extension should be “.jpg” To process multiple input files, the filename extension should be ‘.cmd’, which is interpreted as a command file.

PEG ImageConvert verifies the file is truly of the type indicated by the file extension by attempting to read the file header information and verifying the

file header makes sense for the indicated file type. An error is reported if the file header information and filename extension do not correspond.

2.1.2 Compression

PEG ImageConvert can optionally apply a simple RLE compression technique to the output data. The effectiveness of this compression depends on many factors. If your input image is a computer generated image with few colors, RLE compression can be very effective. If your input image was produced with a RAY-tracing package or from an actual photograph, RLE compression is less successful.

When RLE compression is enabled and if you select 256 or fewer colors in the Output Colors field, PEG ImageConvert is required to save the output data in 8-bpp format. This means for 1-bpp, 2-bpp, and 4-bpp input images, turning on RLE compression forces PEG ImageConvert to first expand the image to 8-bpp format, and then apply RLE compression. Depending on the exact image file, this can actually cause the final output file to be larger than if compression is not used.

For this reason, selecting RLE compression is actually only a suggestion to PEG ImageConvert. If RLE compression is effective at reducing image size, the compression is performed. If compression does not reduce the output image size, the RLE encoding is omitted. This decision is made automatically by PEG ImageConvert during the conversion process. Therefore, the only reason to disable RLE compression is if your PegScreen derived screen driver does not support RLE encoded PegBitmap formats. The PegScreen drivers provided with PEG do support RLE encoding.

RLE compression is almost always beneficial if you are using 8-bpp bitmap encoding, especially for very large images. Compression ratios typically vary from 10:1 to 3:2, depending again on the source of the image being processed.

The use of dithering on the output bitmap has an negative impact on RLE compression effectiveness. For this reason, if data size is the most important consideration in your application, you should disable the dithering option. This forces PEG ImageConvert to do a “Best Match” color mapping of input to output colors.

2.1.3 Color Matching Options

PEG ImageConvert can apply various color optimization and dithering methods when converting the input images to PegBitmap encoded data structures. Your input images can be any combination of 2, 4, 16, 256, hi-color (i.e. 16-bpp) or true-color (i.e. 24-bpp) input images. PEG ImageConvert will convert the input images to best possible representations on the target system.

If your source images contain a higher number of colors than are available on your target display, PEG ImageConvert will reduce the number of colors in the source image. This reduction will either perform a best-match remapping or a dithering algorithm, depending on whether or not the dithering option is selected.

For example, suppose you have several full-color GIF images you intend to use on a target system that utilizes a 4 color grayscale LCD display. In this case, you would select 2-bpp output format, and PEG ImageConvert will reduce the full color GIF images to the best possible grayscale representation.

Fixed Orthogonal Palette

The ***Fixed Orthogonal Palette*** option instructs PEG ImageConvert to use a pre-defined palette covering the rainbow of colors available for 16 or 256 color targets. This is the only palette option when running with fewer than 256 colors. When targeting 256-color operation, you must choose between the fixed pre-defined system palette (the ***Fixed Orthogonal Palette***) or an optimal system palette created for your images. The 256-color fixed orthogonal palette used by PEG is contained in the file peg\source\pal256.cpp.

Generate Optimal Palette

The ***Generate Optimal Palette*** option is only available if you are targeting 256 color (i.e. 8-bpp) output. This is the opposite of using a Fixed Orthogonal palette. When this option is selected, PEG ImageConvert will create a custom palette for use with the input images. The custom palette will be saved at the top of the output file, and will be named **PegCustomPalette**. The custom palette is simply an array of 256*3 unsigned characters, which is passed to the PegScreen::SetPalette function when you want to use the custom palette.

Most users prefer to generate and use a custom palette when running in 256-color or higher modes. This provides you with the best possible image display. Since the resulting palette is generally modified extensively as compared to the palette that was included in the input images, the input images are automatically re-encoded by PEG ImageConvert to use the newly created palette. This all happens transparently when the 'Generate Optimal' option is selected in the PEG ImageConvert dialog. You do not have to do anything special when you are creating your image bitmap or GIF files in order to use a custom palette.

The custom palette created by PEG ImageConvert is always named **PegCustomPalette**, and is found at the top of the ASCII output file generated by PEG ImageConvert. This palette always starts with the 16 standard PEG colors, and is followed by up to 240 colors selected to produce the best possible image display for your input images. The custom palette is simply an array of unsigned characters containing the Red, Green, and Blue components of each color. This array of RGB values should be programmed into your video controller palette registers prior to displaying the associated bitmap(s). This palette can be directly passed to the PegScreen::SetPalette() function, as shown below:

```
Screen()->SetPalette(PegCustomPalette, 0, 256);
```

It is also possible to use multiple custom palettes. When multiple custom palettes are used, it is the responsibility of the application level software to install the correct custom palette before the corresponding images are displayed. For systems that display 'one window at a time', it is a simple matter to install the correct palette when each window is displayed. For other systems, it can be complex to use multiple palettes, and one optimal palette is generally preferred.

Custom Palette

The Custom Palette option allows you to define with a simple editor any palette you prefer, and PEG ImageConvert will remap each pixel color value in the input file(s) to your preferred palette before generating the output image. The custom palette editor allows you to edit and save the palette RGB values, and you can save the palette defintions to any number of custom palette files.

Floyd-Steinburg Dither

The ***Floyd-Steinburg Dither*** option instructs PEG ImageConvert to dither your images when re-encoding them to the target palette. Dithering can be

used in any of the output color depths, with or without a custom palette. What is dithering?? You should realize when an optimal palette is created for multiple images, the actual colors contained in the final palette may not exactly match the original image colors. Likewise when PEG ImageConvert is outputting bitmaps for 16-color targets using input images which contain 256 or more colors, PEG ImageConvert must translate those original colors into the best possible representation using only the 16-color palette.

The dithering option tells PEG ImageConvert how to convert your original image colors to the new system palette colors. If dithering is selected, PEG ImageConvert will pick colors such that the average value in each multi-pixel area is equal to the average value of the original input colors for the same multi-pixel area. If dithering is disabled, PEG ImageConvert will simply translate each pixel into its nearest color in the target palette.

Should you dither? It depends on your target system, your palette options, and your input images. If you are creating a custom palette, dithering usually has very little effect since the custom palette will contain colors very close to the original image colors. In most other cases, dithering can significantly improve your color display, especially if you are displaying photographic images on a target with fewer than 256 colors. The drawback to dithering is your images can take on a speckled appearance, especially if large areas of the original image are in a solid color. If your source images are photographic or ray-traced images, you will almost certainly want to dither. If your input images are hand-drawn images with few colors, you may want to disable dithering and use a best-match color mapping.

2.1.4 Screen Rotation

The screen rotation options change the format of the data produced by PEG ImageConvert. These options should be set to “None” unless you are using one of the profile mode screen drivers. Refer to the chapter on PegScreen in the Programming Manual for more information on the profile mode screen driver templates.

PEG ImageConvert normally outputs image data in a left to right, top to bottom format. The first data byte corresponds to the upper left image pixel, the next byte corresponds to pixel (1,0), and so on scanning from left to right across the image and moving from the top row to the last row. This is the format expected by the standard PegScreen classes.

The profile mode screen drivers are able to display image data more quickly if the data is stored in a different format corresponding to the screen

rotation. The “90” rotation setting should be used when your display device has been mechanically rotated 90 degrees counter-clockwise. When this is the case, PEG ImageConvert outputs the image data such that the first data byte is the lower-left pixel of the image. Data is then saved in a bottom-to-top, left to right manner.

The “270” rotation setting should be used when your display device has been rotated 90 degrees clockwise (i.e. 270 degrees counter-clockwise). When this is the case, PEG ImageConvert outputs the image data such that the first data byte is the upper-right pixel of the image. Data is then saved in a top-to-bottom, left-to-right manner.

These output format modifications are completely transparent to your application level software. Your application software simply passes PegBitmap addresses to the PegScreen drawing functions. However, it is important when using the PEG ImageConvert utility you set the Screen Rotation setting to match the display driver you are using.

Note the pre-defined PEG system bitmaps, contained in the file pbitmaps.cpp, are provided in all forms including non-rotated, clockwise rotation, and clockwise rotation. Only the format corresponding to the screen driver in use is actually compiled and linked into the target system software.

2.1.5 Transparency

The Transparency field can be used to specify a transparent color in the input image. This field only applies to MS Windows or OS2 bitmap and JPG files, as GIF and PNG files encode transparency information internal to the input file. If the output color depth is set to 8-bpp or less, the transparent color will be saved as index 255 in the output PegBitmap, since index 255 is always interpreted as transparent by the ***PegScreen*** bitmap functions. In this case, transparency forces the ***output PegBitmap structure*** to use 8-bpp encoding, since the default transparent color value is 255. This is true even if your source image contains only 2, 4, or 16 colors. If the source image is encoded using less than 8 bits-per-pixel, ImageConvert will expand the image to 8 bpp format when you enable transparency. However, if the color depth is set to 16 or 24-bpp, the transparent color will be saved as 1. There will be no need to change the encoding in this case.

MS Windows bitmap and JPG files do not inherently support transparency. To use image transparency, the source image(s) must be created such that

all areas that should be displayed transparently are 'painted' with an otherwise unused color. You must then inform ImageConvert which color should be interpreted as transparent. There are two methods of specifying the transparent color:

Specify RGB Value

If you select the 'RGB' button, you should enter the Red, Green, and Blue values in the string fields which are displayed. You can determine the correct values through examination with a quality paint program.

Use Upper-Left Color

When this method is selected, ImageConvert will assume that the upper-left corner pixel is in the transparent color.

2.1.6 Conversion Format

The conversion format field allows you to specify how the generated PegBitmap structures will be encoded, and tells PEG ImageConvert how many colors are available on the target system. PegBitmap structures can be encoded using 1, 2, 4, 8, 16, or 24 bits-per-pixel. If you are using a system which supports less than 256 colors, saving the output PegBitmap structures in one of these lower bit-per-pixel formats can save a large amount of memory. Note however, using 1, 2, or 4 bpp output formats requires your PegScreen driver class recognizes and properly displays PegBitmaps saved in this format. It is also possible to use a combination of different PegBitmap encodings in one PEG application, as long as your display driver can handle each of the formats in use.

Within the 6 supported color depths, there are multiple formats that are also supported:

256 Color – Normal Palette

This is the default mode for 256 colors, in which each color value is an index into a set of RGB values maintained in the video controller color palette. The palette uses 3 bytes per color, where each byte represents either red, green or blue.

256 Color – Packed 3:3:2

In this mode, each data value passed to the video controller represents 3 bits of red, 3 bits of green, and 2 bits of blue color data. A small set of video controller chips only support this 256 color mode. Note that when running in packed pixel mode, the color palette is not required and is not used.

256 Color – ARGB 4:4:4:4

This mode looks very much like the Normal Palette mode because each color value is an index into a palette. The difference is that the palette contains 2 bytes per color instead of 3. The structure of the colors in the palette is 4 bits for the alpha channel, 4 bits for red, 4 bits for green and 4 bits for blue.

HiColor – Standard

This is the default mode for 16-bpp. Each color value is a packed PEGUSHORT where 5 bits are red, 6 bits are green and 5 bits are blue. No palette is needed because each color value is an actual RGB value instead of an index.

HiColor – Format 5:5:5

This mode is very similar to the standard HiColor mode, with the exception that there are only 5 bits of green. The most significant bit is therefore unused in each color value.

HiColor – BGR Order

In this mode, the red and blue sections of each color value are swapped. A small number of video controllers require this format. This can be used with either the standard 5:6:5 or 5:5:5 mode.

TrueColor – Standard

This is the default mode for 24-bpp. Each color value is 3 bytes, using 1 byte for blue, 1 byte for green and 1 byte for red. No palette is needed because the color values are the actual RGB values instead of indices.

TrueColor – RGB Order

In this mode, the red and blue sections of each color value are swapped. Different video controllers support different ordering of bytes, so PEG+ supports both.

If the output color depth is set to 256 colors or less, and transparency or RLE compression are enabled, the resulting PegBitmap structures are saved using 8-bpp encoding, regardless of your selection in the Conversion Format field.

2.1.7 Batch Conversion

PEG ImageConvert can be used to perform list or batch conversion of multiple images. This is helpful in almost all cases, and absolutely essential when the goal is to create an optimal palette for multiple images. Batch conversion is selected by specifying an input file with a filename extension of '.cmd', which indicates the source file is actually a command file, rather than an individual image file.

Command files are simply lists of input images, along with optional comments. They do not instruct PEG ImageConvert in terms of the type of conversion to perform. This is still done by selecting the appropriate options in the PEG ImageConvert dialog.

The command file should be an ASCII command file, with one command per line. Each command line should contain a single input image path and filename, and the output image name. The source file and output image name can be separated by any combination of space, comma, and tab characters. The output image name is the name you will use in your application software when passing the image address to one of the PegScreen bitmap display functions.

When performing batch conversion, all of the resulting output images are saved in one output file, along with the custom palette if a custom palette has been generated.

Very large command files are often easier to maintain by entering comments within the file to indicate where groups of bitmaps files are used. Comment lines are indicated by a single '#' character in the first column of a line.

The following is an example of a typical command file:

```
#
# This is a comment line
# Each command line specifies one input file,
# and the name of the resulting PegBitmap structure.
#
\graphics\bitmaps\stop.bmp   StopSign
\graphics\bitmaps\go.bmp     GoSign
#
# note that .bmp and .png files can be processed
# within the same .cmd file
#
\graphics\pngs\yield.png     YieldSign
```

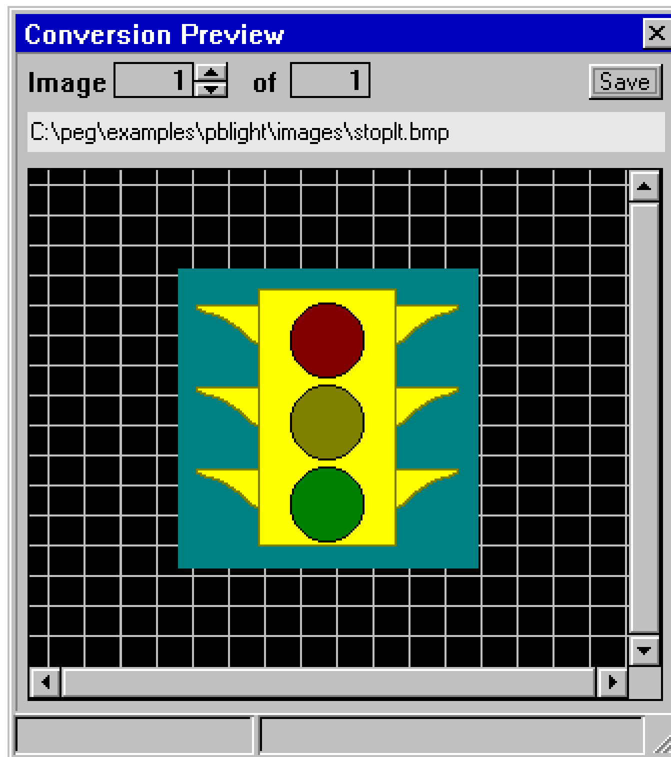
In the above example, the goal is to produce three PegBitmaps and a single optimal palette for use with these three images. The source images are ***stop.bmp, go.bmp, and yield.png***.

The resulting PegBitmaps will be named ***gbStopSignBitmap, gbGoSignBitmap, and gbYieldSignBitmap***, respectively.

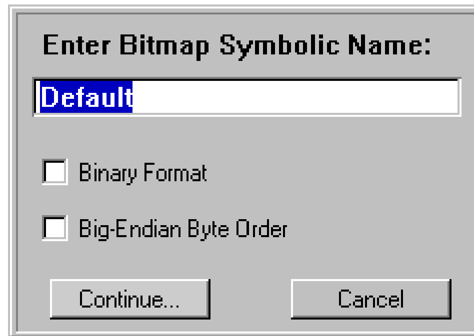
PEG ImageConvert will process each of the input files using the options selected in the PEG ImageConvert dialog, and will save all output to a single file.

2.1.8 Conversion Preview

When the Convert button is pressed, PEG ImageConvert converts the image(s) into PegBitmaps and then displays the Conversion Preview Dialog, which shows all of the newly generated PegBitmaps. If there is more than one image, a spinbutton can be used to cycle through them all. The Conversion Preview Dialog is displayed below:



When the Save button is pressed, another dialog will appear with a few more output options:



Bitmap Symbolic Name

This field only appears if you are converting a single image file, rather than a collection of images in a command file. It defines the name of the PegBitmap structure that will be written to a file. This isn't necessary when using a command file, because the symbolic names are already included in the command file itself. By default, the symbolic name field contains the string "Default", which would produce a PegBitmap name of gbDefaultBitmap. This can be changed to any string that would be valid as a variable name in 'C' source code.

Binary Format

The Binary Format option specifies whether PEG ImageConvert should generate 'C' source code or binary data. If your target system has means for file I/O, you can greatly reduce the RAM or ROM storage requirements for your bitmaps by saving them as binary files, and retrieving them from the file system when they are needed. Binary data can only be used if your final target system supports means for file I/O.

When 'C' source data structures are generated, PEG ImageConvert writes a normal ASCII file that can be opened and modified with any editor. This ASCII file contains the bitmap data array, along with the corresponding PegBitmap structure definition. If an optimal palette is generated, the ASCII file will also contain the custom palette.

When binary format is selected, PEG ImageConvert generates a binary data file containing one or more PegBitmap data structures and bitmap

data definitions. The binary file starts with an eight byte leader, shown below:

// Leader, one occurrence per binary image file:

```
char cVersion[4]            // four byte version string, '1.00'
PEGUBYTE uReservedA         // 1 byte reserved
PEGUBYTE uHavePalette       // 1 byte palette flag
PEGUBYTE uReservedB[2]      // 2 unused bytes
```

The uHavePalette byte of the leader signals the presence or absence of a color palette in the binary file. If the binary file contains a custom palette, uHavePalette will be non-zero and the custom palette immediately follows the file leader, and can be declared as shown below:

// Palette, max one occurrence per binary image file

```
PEGUBYTE Palette[256*3] // only present when custom palette is
                        // generated.
```

Following the short leader and optional palette data are the bitmap header and bitmap data fields. The bitmap header and data fields are repeated for each image contained in the file. Each bitmap is contained in the binary file as shown below:

```
// repeated for each bitmap in file:

char cName[28]              // 28 bytes, bitmap name left-
                            //   justified, padded with NULLS
PEGUBYTE uFlags             // 1 byte, PegBitmap.uFlags
PEGUBYTE uBitsPerPix        // 1 byte, PegBitmap.uBitsPix
PEGUSHORT wWidth            // 2 bytes, PegBitmap.wWidth
PEGUSHORT wHeight           // 2 bytes, PegBitmap.wHeight
PEGUSHORT wReserved         // 2 bytes, unused
PEGULONG lSize              // 4 bytes, size of data array in
                            //   bytes
PEGUBYTE uReserved          // 1 byte, unused
PEGULONG lTransColor        // 4 bytes, PegBitmap.dTransColor
PEGUBYTE uData[lSize]       // bitmap data values
```

For each bitmap, lSize bytes of bitmap data immediately follow the bitmap header information. When multiple PegBitmaps are generated using batch conversion, each successive bitmap header immediately follows the previous bitmap data. There are no padding or alignment bytes inserted between bitmaps.

For multi-byte fields such as wWidth and wHeight, byte swapping may be required when reading the bitmap header data depending on the endian type of your CPU and the method used to read the bitmap data value. PEG ImageConvert always writes multi-byte values MSB (most significant byte) first in binary mode. Byte swapping is not required for the actual bitmap data, as this section always contains only single-byte values.

Reading a bitmap or series of bitmaps from a binary file can be accomplished with the pseudo-code shown below. Several variations of this example could also be used:

```
PEGUBYTE *ReadBitmap(FILE *pSrc, PegBitmap &Bitmap)
{
    PEGUBYTE uTemp[30];
    PEGUBYTE *pPalette;
    PEGULONG lDataSize;

    fread(uTemp, 1, 8, pSrc); // read in the leader

    // ** check here for correct version string **

    if (uTemp[5]) // does file contain a palette?
    {
        pPalette = new PEGUBYTE[256 * 3];

        // read the palette
        fread(pPalette, 1, 256 * 3, pSrc);
    }
    else
    {
        pPalette = NULL;
    }

    fread(uTemp, 1, 28, pSrc); // read image name

    // ** verify image name **

    fread(&Bitmap.uFlags, 1, 1, pSrc);
    fread(&Bitmap.uBitsPix, 1, 1, pSrc);
    fread(&Bitmap.wWidth, 1, 2, pSrc);
    fread(&Bitmap.wHeight, 1, 2, pSrc);
    fread(uTemp, 1, 2, pSrc); // skip unused bytes
    fread(&lDataSize, 1, 4, pSrc); // get data size
    fread(uTemp, 1, 1, pSrc); // skip unused byte
    fread(&Bitmap.dTransColor, 1, 4, pSrc);

    // get RAM for bitmap data
    Bitmap.pStart = new PEGUBYTE[lDataSize];

    fread(Bitmap.pStart, 1, lDataSize, pSrc);

    return pPalette;
```

```
}
```

Note that if the binary file contains multiple bitmaps, the application software could also pass to ReadBitmap() the name of the image file to load. In that case, ReadBitmap would loop through the binary images until the correct bitmap name is found.

Big-Endian Byte Order

The other output option available is to write the images in Big Endian format. This only affects 16-bpp PegBitmaps because they use 16-bit `PEGUSHORTs` for their color values, while all other types use single bytes.

When all the options are selected, pressing the Continue button will open the Save File dialog. Once you select a directory and filename, the output will get written to the file.

2.1.9 Implementation Notes

PEG ImageConvert uses the PEG library classes PEG ImageConvert, PegGifConvert, PegJpgConvert, PegBmpConvert, PegPngConvert and PegQuant to do the work of converting your graphic files into the PegBitmap format and producing optimal color palettes. Since these classes are included with the PEG library, you can also create PEG application programs which read and decompress these common graphic file formats at run time. For most embedded applications, run-time decompression is not desired due to the performance and memory requirements. These systems are better served by using PEG ImageConvert to process the application graphics prior to compile time.

The benefit to run-time decompression is of course memory savings. If your application uses a large number of large graphics files, you may decide to use the PEG image conversion classes directly in your application software to perform run-time graphic file decompression.

CHAPTER 3

PEG WINDOWBUILDER™

3.1 Overview

PEG WindowBuilder™ is a rapid prototyping and design tool used to quickly create your PEG windows and dialogs. PEG WindowBuilder is normally provided as a Win32 application program, and will therefore run on nearly any PC running XP MS Windows '95, '98, 2000, or XP MS Windows NT. An alternate version of PEG WindowBuilder suitable for use in an X11 development environment is also available and provided to users who specify this development environment.

While PEG WindowBuilder may at first glance appear to be a normal MS Windows application, PEG WindowBuilder is actually a PEG application program, running in our Win32 (or X11) development environment. There are several reasons why PEG WindowBuilder is written entirely using the PEG library. First, PEG WindowBuilder is a relatively complex application program and therefore presents a good test case for insuring the PEG library provides the features and capabilities required by complex programs. Second, since the entire interface is created using PEG library classes, exact WYSIWYG appearance is absolutely guaranteed. The appearance of your PEG objects created from within PEG WindowBuilder will match exactly the appearance of those same objects on your target system, within the normal variations of pixel size, aspect ratio, and color performance. Finally, since PEG WindowBuilder is based primarily on the PEG library, it is possible to port and run WindowBuilder on non-PC platforms.

The main PEG WindowBuilder window can be re-sized to take full advantage of your screen real estate. In this case, the virtual frame buffer that is always used by PEG under Win32 is re-sized to match the client area of the outer window. This allows you to see your target windows and dialogs at full size.

3.1.1 PEG WindowBuilder Project Files

All the work you do while running PEG WindowBuilder is saved in a binary data structure called your PEG WindowBuilder ***Project***. This data structure,

when saved to disk, is your PEG WindowBuilder Project file. PEG WindowBuilder project files have the extension ".wbp".

The project file accurately maintains information about the source files, target system, images, strings and fonts, etc.. used by your application program. You can save your work at any time, and later re-open the project file and modify your target screens.

Project Path Information

All project file path information, such as the location of image files referenced by your project, is maintained in a relative path format. This means you can easily copy your PEG WindowBuilder project files from one computer to another as long as you also copy all related font and image files and maintain the same sub-directory structure for your project (if any) in all cases.

Optionally, if PEG WindowBuilder does not find a required image or font file using the relative path information, PEG WindowBuilder always attempts to find the file in the directory containing the PEG WindowBuilder project. This makes it possible to "package" a project and the supporting image and font files in any common directory. PEG WindowBuilder will find the image and font files even if the relative path information is incorrect, or if the file resides in the same directory as the project itself.

3.1.2 Source Output Files

The goal of PEG WindowBuilder is to produce C++ source files, ready to compile and run on your target system. All of the layout, property settings, images, fonts, etc., you use while running PEG WindowBuilder will at some point be exported in the form of C++ source files.

These source files contain standard C++ source code that, when compiled and linked with the PEG library, can be run on your target system. These source files may be one of three types: C++ class definitions, image file data structures, or string table data. Each C++ source file created by PEG WindowBuilder has the extension '.cpp'.

For most .cpp source files, PEG WindowBuilder also creates a corresponding header file. These header files contain class prototypes, message definitions, control IDs, string IDs, and other definitions required for your application software to compile and run.

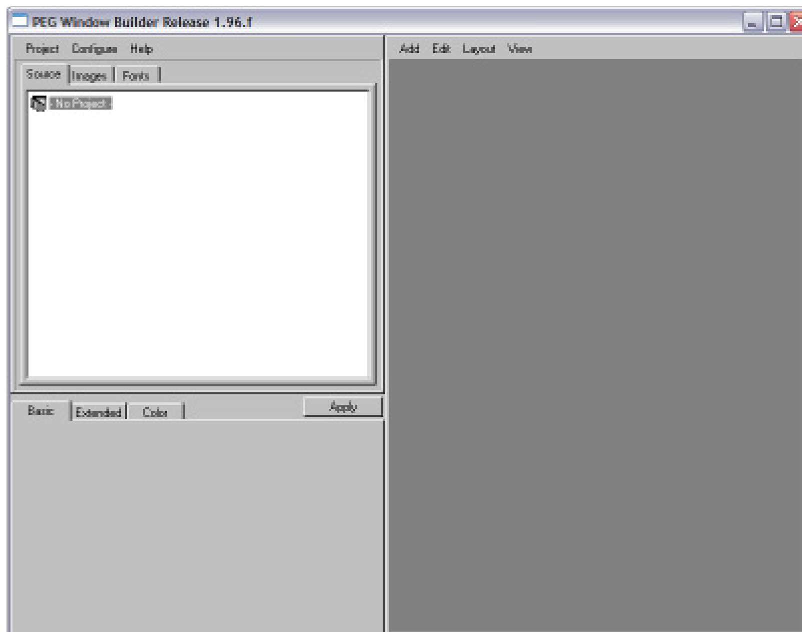
Each top-level module in your PEG WindowBuilder project will generate one C++ source module and one corresponding header file. These files contain the PegWindow derived classes which constitute your application screens.

The source code created by PEG WindowBuilder contains both the object initialization code required to create the window or dialog under construction, and the message handling functions required to process any signals enabled for individual controls. While it is your job to insert the actual signal handling code, PEG WindowBuilder creates a framework for you complete with the individual signal case statements.

If you use BMP, GIF, JPG, or PNG images in your project (described in detail in a later section), PEG WindowBuilder will also produce a single image file containing all images added to the project. This is a 'C' format source file containing data structures defining each of the .bmp, .gif, .jpg and .png images you have added to your project and will be using on your target system.

3.1.3 Screen Layout

When you run PEG WindowBuilder for the first time, you will see the screen shown below. This is the default appearance of the PEG WindowBuilder application.



The PEG WindowBuilder environment contains three main windows. These windows are the **Project** window, the **Properties** window, and the **Target** window. The **Project** window is where you can modify global configuration settings, maintain the source files included in the open project, and command PEG WindowBuilder to perform various operations affecting the entire application program. The **Properties** window displays various properties and style settings for a selected graphical element. This window changes to the “preview mode” when the Project tree Images or Fonts tab is selected, providing a preview of the selected image or font. The **Target** window provides true WYSIWYG emulation of the target system display screen.

Each of these three main PEG WindowBuilder windows will be described in detail in the following sections.

3.2 The Project Window

The PEG WindowBuilder project window is where all information global to your project is maintained. A project consists of any number of source files and associated classes, along with fonts, images, and strings used by your system. While it is possible to use multiple project files for a single system, this is discouraged since there is no way in this case for PEG WindowBuilder to prevent the duplication of source code or data.

The Project Window ***Source*** view displays a tree structure of your top-level windows and each of their child controls. This is the default view you will want to use while creating and editing a new window. To modify the properties of any graphical element, you can select the element either in Project Window Source view, or, if the element is visible you can select the element directly by clicking on it in the Target screen window.

Internally, the Project window correlates directly to and continuously updates the PEG WindowBuilder project file, which has the file extension **‘.wbp’**, which stands for **W**indow**B**uilder **P**roject. The project file contains information about the source files included in your project, the class names and types contained within each source file, the fonts used by your project, etc. Many operations (for example changing the target screen resolution), in addition to affecting the Target window, are also recorded in the **‘.wbp’** project file. The project file is not used or included in your system software, but is used only by PEG WindowBuilder. Still, your project file is so important we recommend you make a backup or tagged version of your

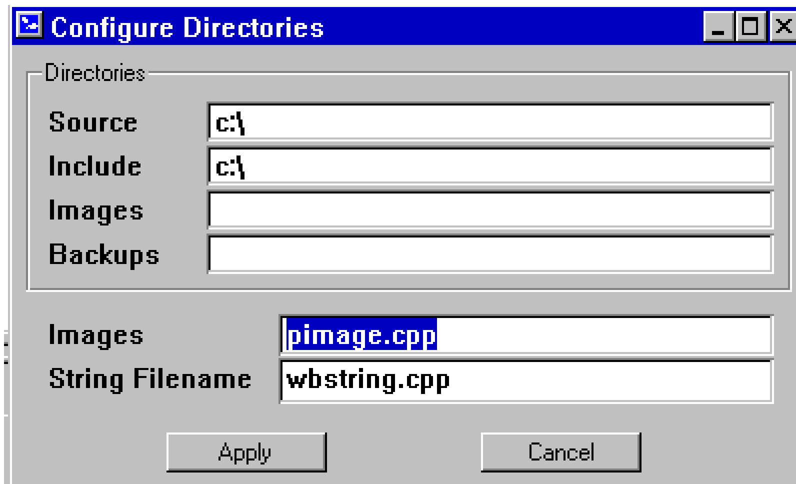
PEG WindowBuilder project file every time you make a tagged version of your system software.

The Project window contains a command menu, along with a PegNotebook control. The PegNotebook tabs are titled *Source, Images*, and *Fonts*. We will begin by describing each of the project window menu commands, followed by a description of each of the notebook pages.

3.3 Project Window Menu Commands

Configure|Directories

This command causes the following dialog window to be displayed:



This dialog window allows you to specify the complete path for your source files, include files, and image files. The **Source Files** directory indicates where the source files generated by PEG WindowBuilder will be saved. The **Header Files** directory indicates where the header files generated by PEG WindowBuilder will be saved. This directory can be the same directory as the source files directory if desired.

The **Image** directory indicates where the image file produced by PEG WindowBuilder should be placed on your hard drive. The name of this image output file is specified in the **Image File Name** field. The BMP, GIF, PNG, and JPG images you incorporate in your project may reside at any location on your system or network; however, for ease of transferring PEG

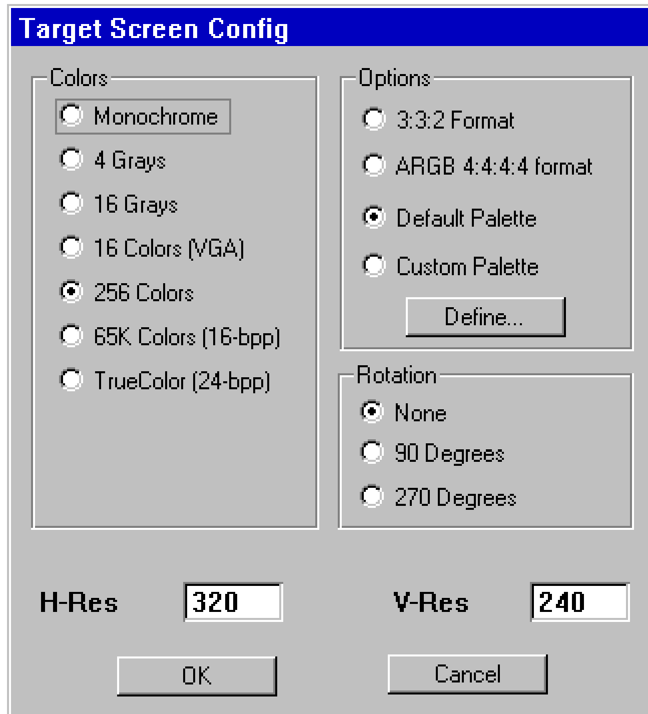
WindowBuilder project files, it is usually best to place all of your project images into a common directory.

The **String Filename** field allows you to specify the name of the PEG WindowBuilder output file which will contain your project string data. There are actually two output files for string information; one contains the literal string table, and the other is a header file containing the string IDs for each string. Both of these files will have the filename specified in the **String Filename** field, however the literal string table will have the extension .cpp and will be saved in the **Source** directory, while the associated header file will have the extension .hpp and will be saved in the **Include** directory.

The **Backup** directory indicates where PEG WindowBuilder will save backup copies of files before updating them. Backups are created for all source, image, string, font, and project files. To disable file backups, set the Backup directory to a NULL string. It is NOT recommended you disable file backups.

Configure|Target

This command invokes the following dialog window, which is used to control the appearance of the **Target** window:



The H-Res field allows you to specify the horizontal resolution, in pixels, of the target screen.

The V-Res field allows you to specify the vertical resolution, in pixels, of the target screen.

The color-depth selections are used to configure the appearance of the **Target** window. You can select 1, 2, 4, 8, 16, or 24-bpp. In addition, for 16-color screens you can specify either color or grayscale appearance, and for 256-color sreens you can choose either a palette mode, a packed 3:3:2, or ARGB 4:4:4:4 mode.

Configure|Languages

This command invokes a dialog window that allows you to specify the number of languages supported by the system software and the character encoding used for string storage. This in effect determines the layout of the String Table that will be generated by PEG WindowBuilder, and determines how strings will be entered when new objects are created that display text or string data.

You will be prompted to select the language associated with each column of the string table. You choose a language by selecting the appropriate 2-letter language abbreviation from the available list. For instance, if you wanted English and German languages in your string table, you would select “en” for the first language and “de” for the second. The first language is the default language used at the start of your PEG application.

If your application does not require support for multiple languages, or if, for any reason do not wish to maintain your string data in the PEG WindowBuilder string table, you can de-select the “Enable String Table” checkbox on the language configuration page.

Configure|Remote

This command allows you to display the contents of the target screen window in a secondary Microsoft Windows window. This feature is used when the target output device can be driven by a second video card installed in the PC on which PEG WindowBuilder is running. Under certain Windows platforms, a second video card is utilized as an ‘extended desktop’. This capability allows you to drag the remote view window to the second video display, and view your screens on the target display as you create them.

Note the remote view window is limited to drawing only the portion of the target screen which is visible in the target window. This means you must resize your WB window large enough to see the entire target screen in order to see the full window on the second display device.

Project|New

This command creates a new PEG WindowBuilder project file. The project file will initially contain only the default image file, and will use the default Target window configuration.

Project|Open

This command is used to open a previously saved PEG WindowBuilder project file.

Project|Open Recent

This command is used to open a recently edited project file.

Project|Save

This command is used to save the current project file. Note the associated source files are not updated, rather only the binary project file is saved to disk.

Project|Close

This command is used to close the current project file. If the project file has been modified you will be prompted to save your changes before the project is closed.

Project|Add Module

This command adds a new source module to the current project. One or more source modules must be added to a new project before you will be able to edit anything using the target screen menu within PEG WindowBuilder.

Project|Import Module

This command allows you to import a source module from a second WindowBuilder project. This facility is used to merge project files created by separate developers working on a common project.

Project|Add Image

This command adds a new image to the current project. Once an image has been added, it can be applied to any PEG objects which support bitmap images. The source for the image must be a .bmp (Windows or OS2 Bitmap) .gif (CompuServe GIF), .jpg (JFIF/JPEG), or .png (Portable Network Graphics) file.

The Images page of the notebook must be selected for this command to be active.

Project|Copy Selected Module

This command instructs PEG WindowBuilder to make an exact copy of the currently selected module and add it to the project. The user will be prompted to specify the new file and class name for the new module.

Project|Add Font

This command is used to add any number of custom fonts to your project. Once a font is added, it can be applied to any text-related PEG object simply by dragging the font from the preview window to the object that

should be assigned the custom font. The input for adding a font should be a font file created with the PegFontCapture utility program.

The Fonts page of the notebook must be selected for this command to be active.

Project|Generate Source|Current Source Module

This command instructs PEG WindowBuilder to update the current selected source and include files to reflect changes in the current project. Before the source files are updated backup copies are saved to the Backup directory, unless this directory has been set to NULL.

Project|Generate Source|All Modified Source Modules

This command instructs PEG WindowBuilder to update all source modules that have been modified. Before the source files are updated backup copies are saved to the Backup directory, unless this directory has been set to NULL.

As you work within the Target window, the internal copy of the PEG WindowBuilder project file is updated to reflect all of your changes. The source code files for your project are NOT updated until the Project|Update|Source command is invoked.

Project|Generate Source|Generate Image File

This command instructs PEG WindowBuilder to re-process all input image files, and save the resulting PegBitmap information to the **Image File Name** in the source directory. Note the output image file is completely regenerated each time an image update is performed. It is therefore not advisable to manually edit the output image file, since any changes will be overwritten by an image update.

Project|String Table|String Table Editor

This command brings up the string table edit window. This window allows you to define the literal strings and string IDs used for each language in the system. The string table is further described in a following section entitled **The String Table**.

Project|String Table|Generate String File in Source Form

This command instructs PEG WindowBuilder to re-create the string data table and associated string ID header file. Note this file is completely regenerated each time the Update Strings command is issued, and it is therefore not advisable to manually edit the generated string data files.

Project|String Table|Generate Binary String Resource File

This command instructs PEG WindowBuilder to save the string table and fonts into a binary string resource file that can be loaded into an application at runtime.

Project|String Table|Export as Unicode Text File

This command instructs WindowBuilder to export the string data in a Unicode text file format that can be opened and edited by any external program that supports the Unicode text file format.

Project|String Table|Import Unicode Text File

The command instructs WindowBuilder to read a Unicode text file and import new string translations from the imported file. When executing this command, WindowBuilder searches the imported file for StringID names that match existing entries in the string table. When matching StringID names are found, the literal string entries for each language are imported into the WindowBuilder string table. Note the Unicode text file must be in the exact format produced by WindowBuilder during the export operation for this command to be utilized correctly. There is no standard format for Export and Import of string data in Unicode text file format.

Project|String Table|Export as XLIFF File

This command instructs PEG WindowBuilder to export the string data in the XLIFF format that can be opened and edited by any external program supports the XLIFF file format. XLIFF is a standard translation file format based on XML. XLIFF is a primarily bi-lingual format, so you can only export either one or two languages at a time.

Project|String Table|Import XLIFF File

This command instructs PEG WindowBuilder to read an XLIFF file and import new string translations from the imported file. When executing this command, WindowBuilder searches the imported file for StringID names

that match existing entries in the string table. When matching StringID names are found, the literal string entries for each language are imported into the WindowBuilder string table. Note that the XLIFF file must be in the exact format produced by WindowBuilder during the export operation for this command to be utilized correctly.

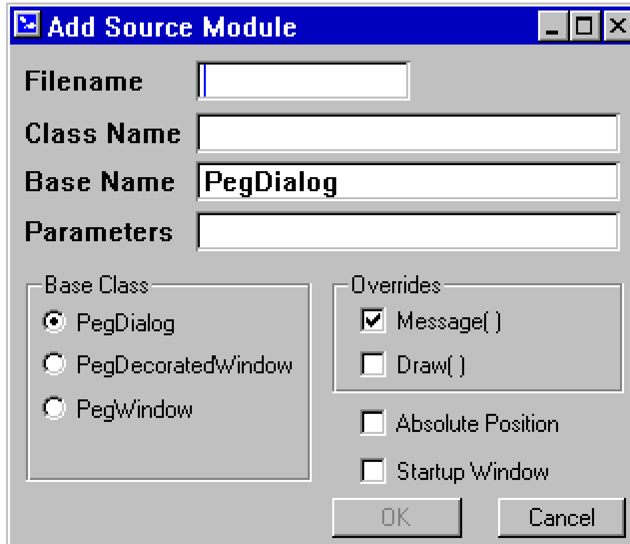
3.3.1 Working with Modules- The Source Page

PEG WindowBuilder organizes your application program, containing possibly hundreds of unique application windows, into unique modules. Each module corresponds to one window or dialog, and produces one source file and one include file.

The **Source** page of the Project window notebook control contains a PegTreeView depicting each of the modules included in the current project. Each top-level node of the PegTreeView control represents a top level class constructed with PEG WindowBuilder. If you expand a top-level node, you will see the list of child objects that have been added to that module. If you click on one of the child objects, that item will become selected inside the Target window.

The Target Window always operates on the selected module. If no module is selected, none of the Target Window editing commands are operational. ***Therefore the very first thing you must do after creating a new project is add at least one module to your project.*** How to do this is described below, but it is important to remember the right-hand side of the program window, the target window, is not operational until a module has been added to the project and selected.

You can at any time create a new module by selecting the Project|Add Module command. You will be presented with a dialog window shown below, prompting you to enter the required information, after which the new source module will be added to your project.



The ***Filename*** field allows you to specify the output filename for the source file PEG WindowBuilder will generate for this new module. Any valid filename may be entered into this field. You do not need to specify an extension, as PEG WindowBuilder will automatically write both a .cpp and a .hpp file for this module.

The ***Class Name*** field allows you to specify the name of the new window class you are creating.

The ***Base Name*** field allows you to specify what the direct base class of the window is going to be. WindowBuilder recognizes 3 standard base classes for top-level modules: PegDialog, PegDecoratedWindow and PegWindow. However, if you are developing screens that are going to be based on some other custom window class, you can put the name of that class here. This will not affect the operations within WindowBuilder, but that name will be used when the source code is generated.

The ***Parameters*** field allows you to specify any user-defined parameters you would like to pass to the class constructor (in addition to the parameters PEG WindowBuilder will always pass to the constructor). If you desire to pass extra parameters, you should type them on this line exactly as they should appear in the constructor prototype, i.e. Type-Name, Type Name, etc., for each parameter.

The ***Overrides*** group is used to tell PEG WindowBuilder which function of the base class will be overridden by the class you are defining. Only two options are supported by PEG WindowBuilder (although you can of course add your own function overrides to the completed class). These are the Message function and the Draw function, which you know by now as the most commonly overridden of all PEG member functions. The default setting of this field indicates that you will override the Message() function (to catch signals from child controls) but will not override the Draw() function. This is the most common situation.

The ***Absolute Position*** checkbox allows you to use an alternate form for the class definition. Normally, PEG WindowBuilder produces a class that accepts a left-top corner position as the first two incoming parameters. The window and all child controls are positioned relative to this left-top position. If desired, you can produce a class that is absolutely postitioned, i.e. there is no left-top incoming parameters and the window are child controls use absolute pixel positioning.

The ***Startup Window*** checkbox specifies this window will be the first displayed when your application executes. When this checkbox is selected, PEG WindowBuilder automatically writes the PegAppInitialize function in this module such that this window is created and added to PegPresentationManager during program startup.

When you have completed entering in the required information, a new object of the selected type is created and displayed in the **Target** window, and the new module is added to the source page tree view control.

To remove a source module and its associated objects from your project, select the source module in the tree control and press the ‘Delete’ key on your keyboard. Following confirmation, the source module is removed from the current project. Note the actual source files corresponding to the selected node are NOT deleted from your hard drive. PEG WindowBuilder simply removes all information about the source file from the current project.

To modify the parameters associated with a source module after the module has been created, you can right-click with the mouse on the module in the Source notebook page. This will bring up a dialog window allowing you to change the module name, file name, and other module parameters.

3.3.2 Working with Images- The Images Page

The second tab of the Project window notebook is named “Images”. The Images Notebook page lists the BMP, GIF, PNG, and JPG image files included in the current project. Similar to the Source page, this information is presented in a PegTreeView. Each top level node of the tree is one PegBitmap image that can be used in your application program.

Images are imported into your project by using the Project|Add Image command on the menu bar while viewing the Images notebook page. Images can be selected from any location on your local hard drive or network drive. PEG WindowBuilder will maintain relative path information to the image if the image is located on the same drive as the project file, allowing you to easily move entire projects from one computer to another.

When an image is selected with the mouse or keyboard on this notebook page, a preview of the image is shown in the preview window. The image can be applied to a bitmap-based PEG control by dragging the image from the preview window to the target control.

You can display any image on the image page by selecting the image, or by using the up-down arrow keys to move up an down in the tree control. Each image is displayed in the Preview window as it is selected, allowing you to quickly scan through the images included in your project.

Images are deleted by selecting the image in the Image notebook page and pressing the 'Delete' key. Any objects that had been using a deleted image are re-configured to use a default bitmap.

When the Target window is being used to layout a new window or dialog, PegBitmap images from the Images page can be directly dragged-and-dropped onto PEG objects that support image display. For example, if you have created a PegBitmapButton as a child of the current window or dialog, you can simply drag a PegBitmap from the images preview onto the PegBitmapButton. This action causes the PegBitmap to be assigned to the PegBitmapButton***. It is important to remember before you can drag an image and drop it in the target window, you must add at least one child object to the target window which is capable of displaying an image.***

When you assign an image to certain types of objects by clicking on the image and dragging it to the object, PEG WindowBuilder will ask you if you would like to re-size the object to fit the image. If you select yes, the target

object is re-sized such that the image fits neatly within the object. If you select NO, the image is centered within the client area of the target object and the target object size is not modified. For other object types the resizing is done without question since this is the only mode of operation supported by that object type.

The operation of PEG WindowBuilder when generating the output image file can be quite complex, depending on your target screen color resolution. Take a deep breath before continuing!

If your target system supports 256 colors, PEG WindowBuilder will scan each image in the project, create an optimal palette for displaying those images, remap the image colors back to the optimal palette, RLE encode each image, save the custom palette, and save each newly-encoded image file in 'C' style source data structures.

For 16 color targets, the Update Images command is slightly less complex. In this mode, WindowBuilder dithers each image to a fixed orthogonal 16-color palette, RLE encodes the images for which this is memory efficient, and saves the resulting bitmaps in 'C' style source data structures.

For 2 and 4 color targets, the Update Images command simply saves each image in the selected format with optional dithering.

3.3.3 Working with Fonts- The Fonts Page

The Fonts page is very similar to the Images page. This page lists the fonts which have been added to the project, and allows you to drag-and-drop fonts onto specific objects.

When you first create a new project you will find two fonts listed on the Fonts page. These are the **System** font and **Menu** Font. These fonts are part of the PEG library, and are always available for you to use. Note you are not allowed to delete these fonts from your project.

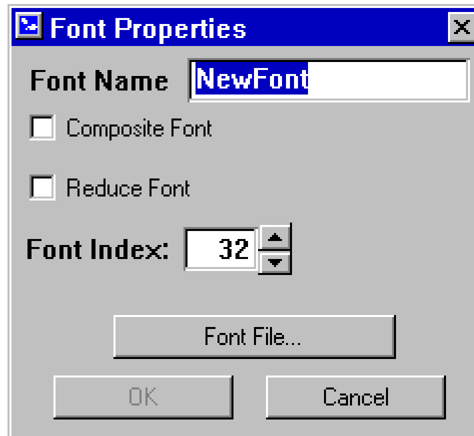
You can also add any number of additional fonts to your project and apply them to any text-display gadget that is part of your interface. You must pre-generate the fonts you will add to your project by using the FontCapture utility program. PEG WindowBuilder is able to read the 'C' source files produced by FontCapture and create PegFont data structures in memory using these source input files. The result is you see exactly the same appearance for your fonts as you will see on your target system.

If you delete a font from your project which has been assigned to one or more text objects, you will receive a warning indicating the font is being used by one or more objects. If you delete the font anyway, the text objects that were using the font will revert to the default font based on object type.

The operation of adding a new font to your project varies depending on your language configuration settings (described in a later section). If the current project is configured to use only ASCII characters without the String Table, adding a new font is simply a matter of choosing a font file produced by the FontCapture program.

Composite Fonts and Reduced Fonts

If your project is configured to support multiple languages and Unicode, the operation of adding and defining a font becomes more complex. This is due to several factors, primarily the large number of characters (> 30,000) which may be required for the support of multiple languages. For projects of this language configuration, the following dialog is displayed when you select the Project|Add Font command:



The ***Font Name*** field assumes the name of the source font for standard (i.e. non-Composite, non-Reduced) fonts. For Composite or Reduced fonts, PEG WindowBuilder will produce a completely new PegFont from the input font data when the String Table file is generated. In this case, you can assign any name to the new font by typing into the Font Name field.

Composite Fonts are fonts collections, produced and managed by PEG WindowBuilder, containing multiple sub-fonts produced by the FontCapture

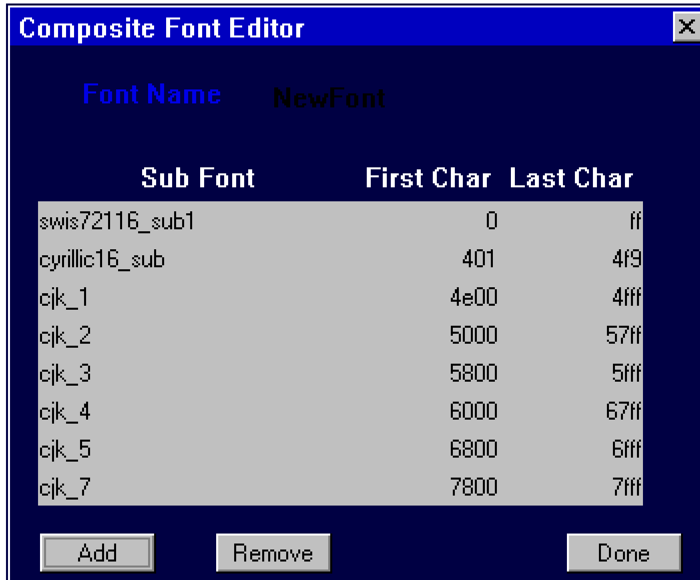
program. Why are composite fonts needed? To overcome limitations in the character set or alphabet included in most TrueType or BDF font files. In many cases you will find it is impossible to obtain one single TrueType or BDF font that contains all of the characters required by your application program. For example, one TrueType font may contain the Latin and Cyrillic characters, while another contains Kanji and Hangul. It is very rare to find a single font which contains characters for many different alphabets.

In order to avoid re-assigning the font associated with every PEG object when a language change is made, it is desirable to have a single font (or multiple fonts of different sizes, each containing the same character set) which contains all of the characters used by your application program. This is the reason for Composite fonts, you can combine any number of sub-fonts in to one "SuperFont" potentially containing all the characters from every sub-font.

The true power of Composite Fonts is realized when combined with the Reduce Font option. This option instructs PEG WindowBuilder to produce a new PegFont wherein only those characters actually used by your application strings are included in the new PegFont. This information about which characters to include is obtained by examining all strings found in the project String Table (described below).

By using the Reduce Font option, you can save a tremendous amount of ROM storage for your fonts for languages with very large alphabets, such as Asian languages.

If you select the Composite Font option, you can then select the "Component Fonts" button to edit a table which defines each sub-font that will be include in the composite font. This table is shown here:



In the above example, several sub-fonts have been added to the composition font to yield one “Super Font”. The composition font contains characters from the Latin, Cyrillic, and several pages of CJK (Chinese-Japanese-Korean) alphabetic characters.

For each sub-font you add to your composite font, the range of characters used from the sub-font will default to the full range of characters contained in the sub-font. Since it is possible several sub-fonts may contain overlapping characters, you may need to edit the First Char/Last Char ranges displayed in this table so each sub-font provides a non-overlapping range of characters to the final composite font.

When you have completed defining the sub-fonts that will make up your composite font, you simply close this table by pressing the Done button, and after naming your composite font click the OK button on the Font properties dialog.

You can return and re-edit your Composite font settings at any time by right-clicking on the font name in the Font tree display. Note, however, while you can change the component font list for a Composite font, ***you cannot change a previously added non-Composite font into a Composite font, nor can you change a Composite font into a normal***

font. Instead, you must delete the font from your project and re-add the font using the desired settings.

For Unicode enabled systems using Reduced fonts, the Project|String Table|Generate String File in Source Form command does far more than simply write out your strings as C++ string arrays. The following operations take place:

- Scans string tables for all languages, creating global table of required glyphs.
- Re-scans all reduced fonts used by the application, saving only the required glyphs.
- Write new font structures containing only the required characters or glyphs.
- Create C++ wide-string arrays for each language supported.

3.4 The Target Window

The PEG WindowBuilder Target Window displays as accurately as possible a representation of the target system display screen. This representation is completely accurate in terms of pixel placement of graphical objects and colors used by each object. The Target Window does not correct for differences in aspect ratio (i.e. pixel squareness) between your PC screen and your target screen.

When you create objects within the Target Window, you are actually defining new instances of PEG objects. These objects are dynamically constructed and added to the Target Window, and operate just as any normal PEG objects. This is important to remember as you create your windows and dialogs within PEG WindowBuilder, you are creating a working PEG program. You can at any time interact with the objects you have created, just as the end user of your system software will interact with the final system.

The target window becomes active when a source module is selected in the project window. If no source files are included in your project, you must first create a new source module before you will be able to do editing in the target window. When you create a new source module, a default object of the type defined in the new source module is generated and is the initial object displayed in the target window. After this step has been completed

you will be able to use the Target window to modify and/or add children to the initially defined object.

3.4.1 Selecting Objects in the Target Window

Almost all selection and editing of objects in the Target window is done using the mouse. When you click on an object in the Target window, a red border is drawn around the object to indicate that the object has been selected. You can re-size any object by dragging the dark border with the left mouse button held down until the desired size is obtained.

You can move an object by either dragging a selected object with the mouse, or by using the keyboard arrow keys.

Multiple objects can be selected by holding the <ctrl> key down while right-clicking on additional objects. When multiple objects are selected, the selection box expands to contain all selected objects.

3.4.2 Target Window Menu Commands

The target window menu commands always operate on the current selected object. You should select an object or group of objects before selecting one of the menu commands. The Target window menu commands are:

Add|Button

This selection brings up a sub-menu of common button objects. These include:

- PegTextButton
- PegMLTextButton
- PegBitmapButton
- PegDecoratedButton
- PegCheckBox
- PegRadioButton
- PegSpinButton
- PegIcon

Selecting any of these commands adds an object of the selected type to the current selected object. In this case, the current selected object would usually be the top-level window or dialog, or possibly a PegGroup.

Add|Text

This selection brings up a sub-menu of common text display objects. These include:

- PegPrompt
- PegVPrompt
- PegEditField
- PegTextBox
- PegEditBox

Selecting any of these command adds an object of the selected type to the current selected object. In this case, the current selected object would usually be the top-level window or dialog, or possibly a PegGroup.

Add|Indicator

This selection brings up a sub-menu of indicator style gadgets. These include:

- PegAnimation
- PegProgressBar
- PegCircularBitmapDial
- PegCircularDial
- PegFiniteDial
- PegFiniteBitmapDial
- PegColorLight
- PegBitmapLight
- PegLinearScale
- PegLinearBitmapScale

Selecting any of these command adds an object of the selected type to the current selected object. In this case, the current selected object would usually be the top-level window or dialog, or possibly a PegGroup

Add|Slide|Scroll

This selection brings up a sub-menu of slider/scroll bar objects. This list includes:

- PegSlider
- PegVertScroll
- PegHorzScroll

Note adding a Vertical Scroll or Horizontal Scroll using this menu command adds a ***client area*** scroll bar. This is a user-defined scroll bar rather than a scroll bar which acts to scroll the window client area. Normal non-client-area scroll bars are added by adjusting the window properties.

Add|Container

This selection brings up a sub-menu of container style controls, that is controls which are used to contain or group other child gadgets. These include:

- PegGroup
- PegComboBox
- PegVertList
- PegHorzList

Add|Chart

This selection brings up a sub-menu of PegChart derived classes that can be added to the current object. These include:

- PegLineChart
- PegStripChart
- PegMultiLineChart

Add|Window

This selection brings up a sub-menu of PegWindow derived classes that can be added to the current object. These include:

- PegWindow
- PegNotebook
- PegTreeView
- PegTable

- PegSpreadSheet

Custom

This command allows the user to rebuild the executable with their custom objects and will appear in the list when PEG WindowBuilder is rebuilt.

Edit|Properties

This command is now obsolete. This used to bring up the properties dialog of whatever object is currently selected. That properties dialog is now always present in the bottom left corner of the window.

The properties dialog is context sensitive depending on the type of object which has been selected. In general you can adjust the border style, system status flags, and style flags for a given object by selecting each page of the properties dialog notebook control. Many object types have additional settings which can be controlled using the properties dialog.

The properties dialog is also where you specify the text string associated with many object types such as PegPrompt or PegString. For text-based control types, the properties dialog extended properties page includes a field labeled “Initial Text” which allows you to type in a string or, if the String Table is enabled, select the string ID associated with an object. This string ID is a member of the string table maintained by PEG WindowBuilder. You can view and edit the string table by selecting the **Project|String Table** command in the project window.

If you have disabled the use of the PEG WindowBuilder string table in the **Project|Configure|Language** dialog, the String page of the properties notebook allows you to directly enter the ASCII string used to initialize a control.

Edit|Copy

Copies the selected object or objects, including all status and style flags. Only one object can be selected when the **Edit|Copy** command is issued, however that object can have any number of children. When an object such as a PegGroup is copied, and the PegGroup has a number of children, the Group AND all of the group children are copied.

When this command is selected, PEG WindowBuilder automatically changes the selection box to contain the parent of the current object. This

allows you to quickly copy and paste an object into the object's parent, which is the most common operation.

Likewise, you can select an object, copy it, and then select an entirely different object to paste the copy into.

Edit|Paste

This command pastes an exact copy of the copied objects into the center of the selected object. PEG WindowBuilder automatically selects the parent of the copied object as the target for the paste command. You can override this operation by selecting any other parent before selecting the paste command.

Edit|Delete

This command deletes a selected object. A object must be selected to be deleted, but a group of selected objects cannot be deleted.

Layout|Align|Left

Layout|Align|Right

Layout|Align|H-Center

Layout|Align|Top

Layout|Align|Bottom

Layout|Align|V-Center

This group of commands is used to evenly align any number of child controls. Before activating this command any number of child controls should first be selected using the method described above. The above group of commands can then be used to exactly align the group of objects as desired.

Layout|Move To Front

This command adjusts the order in which child objects are added to the parent. The Move To Front command makes the object that last object added to its parent. This is useful for adjusting the tab-order of controls added to a parent window.

Layout|Move To Back

This command adjusts the order in which child objects are added to the parent. The Move To Back command makes the object that first object added to its parent. This is useful for adjusting the tab-order of controls added to a parent window.

Layout|Equal Height

This command evenly adjusts the height for a group of selected objects. The height will be adjusted to the largest height in the group of selected objects.

Layout|Equal Width

This command evenly adjusts the width for a group of selected objects. The height will be adjusted to the largest width in the group of selected objects.

View|Maximize

This command removes the entire lift panel to allow the user to get a full view of the working window.

View|Test Mode

This command places the target window in test mode. In test mode, all of the PEG WindowBuilder windows are hidden, leaving only your newly created window or dialog on the screen. While in this mode, your new window or dialog will operate exactly as on the final target system, although any message processing code you have added to the window or dialog will not be operational from within PEG WindowBuilder.

While in test mode, you will not be able to select and edit objects. You can exit edit mode by closing the window or dialog under test, or by pressing the “Stop” button placed in the lower right hand corner of the screen.

If you have defined a product background image and hotspots (described below), you can click on the product hotspots while in test mode to navigate through your UI screens, fully simulating the operation of your interface within the WindowBuilder environment.

View|Zoom Scale

This command allows you to zoom-in on the target window. This is useful when your target screen is very small and it is easier to do layout and modification in an enlarged view of your target screen.

View|Product Image|Select Image

This command allows you to select a background image to wrap your target screen. This can give you a good representation of the “look and feel” of your final device. Any background image may be selected, however you must take care to insure the background image is scaled correctly to fit the target screen. In other words, the background image should include a pixel-for-pixel screen area. You can adjust the actual position of the WindowBuilder screen display within this background image.

View|Product Image|Edit Hotspots

When a background product image has been defined, this command brings up the hotspot editor dialog. This dialog allows you to define areas on the product which will produce input messages into the PegMessageQueue. A common example would be an product which provides the end user with up/down/right/left and select type nagivation keys. You can use the hotspot editor to define the areas within the background image the are selected to produce each of these input message types.

Hotspots are utilized when you use the View|Test Mode command within PEG WindowBuilder. In this mode, you can click on the product background image hotspots to navigate through your UI. This allows you to fully excercise each screen of your UI design without ever producing source code or compiling to produce your actual PEG executable program.

View|Product Image|Remove Background Image

This command is used to remove a previously assigned product background image.

3.5 The String Table Editor

If you have enabled the use of string tables in the PEG WindowBuilder **Configure|Languages** dialog, PEG WindowBuilder will maintain a table containing all strings, for all languages, used by your application program. In this environment, all PegTextThing derived classes are constructed using StringID information, rather than literal strings. This allows your system software to easily convert between different supported languages.

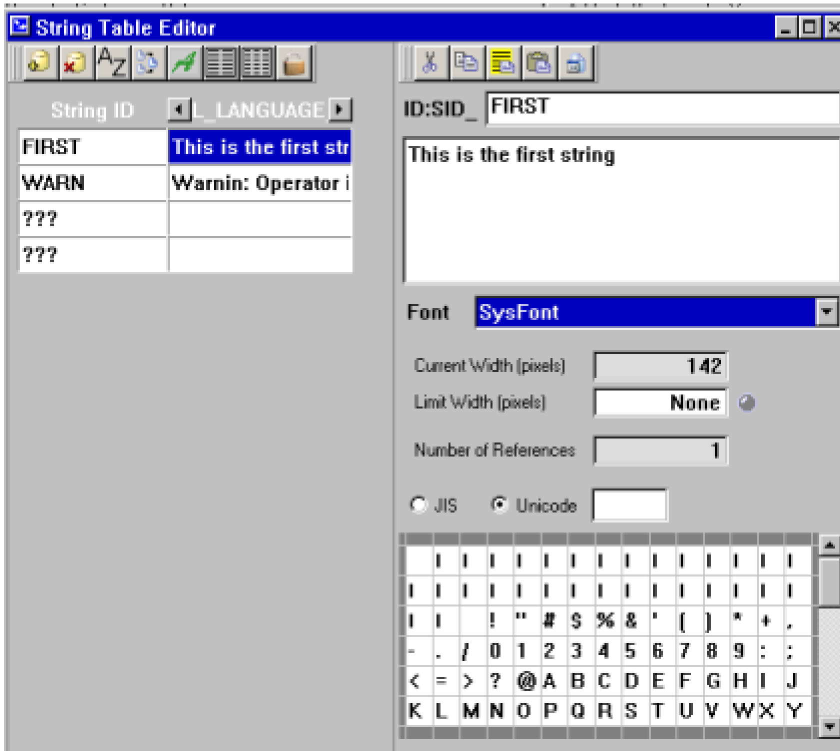
The String Table is composed of an array of literal strings, a two dimensional array of string pointers, and an enumeration of string ID values. String ID values are just indexes selecting the correct row from the two dimensional table. Each table column is associated with one of the

supported languages. If your application supports only one language, the String Table is simply a single list of literal strings, an array of string pointers, and an associated String ID enumeration.

The correct string table column is selected by the current language, which is maintained in a static member variable of the PegTextThing class. This variable should be loaded with one of the enumerated language names when the active language is selected by calling the PegTextThing::SetLanguage() function. The default language is the first language configured in the Configure|Languages dialog box.

The string table is saved to the filename specified in the Configure|Directories dialog. The enumeration of the language names, and the string table IDs, are saved in the corresponding String Table header file. Each source file that uses the String Table must include the String Table header file in order to resolve the string IDs and language names. This include is added to each source file generated by PEG WindowBuilder.

The String Table is edited by selecting the Project|String Table command on the Project window menu bar. This brings up the string table edit window, shown here.



The String Table

The left hand side of the String Table Editor window displays a PegSpreadSheet object containing each of the strings used in your system. Each row of the table corresponds to a StringID, and each column of the table corresponds to a supported language. The enumerated language names are displayed as the table column headers.

The String Table can be displayed in a two-column or three-column format. You can change the format by right-clicking over the spreadsheet and selecting the desired format in the pop-up menu.

You can also sort the string table entries by using the right-click pop-up menu. This menu provides command to shift the selected entry up or down in the string table.

The first language listed on your language configuration page is your project's “Reference Language”. This language will be usually be English,

but may be any language desired. The reference language is important because this is the language you are working in when you work in the target window. This is also the language which is always displayed when you view the table in three column mode.

String Edit Fields

The right-hand side of the String Table Editor window displays a series of fields for editing the selected string. The first field, the ***ID*** field, is where you can modify the string ID name, which is the name associated with each string ID. This name will be included in your string file as an enumerated list, and you will use this name in your application software when you want to refer to a particular string. You can edit this name simply by typing on the keyboard.

You can select the font to use while working in the string table using the drop-down list box labeled ***Font***. Once you select the font to use in the String Table Editor, PEG WindowBuilder remembers which font you have selected each time you call up the editor window. For example, if you have created a composite font supporting all of your languages, you can specify that this font should be used in the String Table Editor. Each time you call up the String Table Editor, this is the font which will be used unless you select a different font.

When you select a font to use in the String Table Editor, the font is displayed in the grid in the lower-right portion of the screen. This grid is more than a display, but actually allows you to select characters while editing the current string. This is required since for many languages you may not be able to simply type string values since the language alphabet contains characters which are not included on your keyboard.

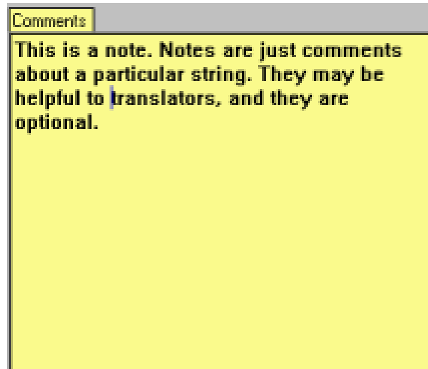
The second field on the right side of the String Table Editor window is the string literal edit window. This field displays the string literal value using any of the fonts which are part of your project.

There are three methods for editing strings displayed in the string literal edit window. First, if the current language alphabet is supported by your keyboard, you can simply type the string value. Second, you can simply click on characters displayed in the font viewer window. As you click on the characters, they are inserted into the current string at the current insertion point. Finally, you can type the JIS or Unicode encoding value for the character you wish to insert. For some people who know the encodings for

common characters, this is faster than finding the characters in the font display window.

As you edit the selected string, the width of the string (in pixels) is displayed in the **Width** field. This can be useful to insure the string for every language will fit properly in the display area in which the string is used.

The **Notes** button brings up a small note editor window, shown here:



Notes are useful for including additional information about each string, usually for the benefit of translators who will translate your English or reference language strings into strings for the other languages.

As noted previously the first language you configure in the Configure| Languages dialog is described as your ***Reference Language***. The reference language is the language you will use while working in the Target window, and it is the language you will always view when the String Table Editor is in three-column mode. The reference language will usually be English, but may be any supported language.

The reference language is also important in the event a translation is not required or available for certain strings in your application. For example, let's suppose the string "California" is included in your application. Since this is a proper name, the name of a State, translation to another language is usually not required. Therefore, you would leave the string blank for every other language except your reference language, English. When PEG WindowBuilder generates your StringTable file, any blank or NULL strings for secondary languages are automatically filled in with the reference language string pointer. In other words, for every other language in your application, the "California" string entry will be filled with a pointer to the

reference English string "California", ***you do not need to duplicate this string for every language***.

3.5.1 Merging String Tables

The ***Merge…*** button on the String Table Editor window invokes a series of dialog that walk you through the merge process. In order to understand the reason for the merge operation, we need to examine the life-cycle of a typical multi-language project development.

Step 1)The system developers define the initial string table. The total number of languages and the language names are defined using the Configure|Languages dialog.

Step 2)The String ID names and the Reference Language (English) strings are initialized for all strings in the application using the String Table Editor.

Step 3)The PEG WindowBuilder Project file, along with the PEG WindowBuilder executable program, are distributed to translators who will each fill in one column of the string table. These translators may reside at the same location, but often reside all around the globe.

Step 4)The translators return PEG WindowBuilder project files to you, and the returned project files each have one or more additional columns of the string table filled in with translated strings.

The problem should now be obvious: how do you get the translated strings from all of those different translators back into a common project file?? Enter the Merge operation. The Merge operation will merge strings for selected languages from a second project file into the current project file. The process is actually very simple as you are guided step-by-step through the merge process. When PEG WindowBuilder performs the merge, it looks for matching string ID names in the secondary project. For each matching string ID name, if the selected language in the secondary project has a non-NULL string value, that string value is copied into the current project for that specific string ID and language.

3.5.2 Exporting the String Table

The String Table data can be written out to a C++ source file at any time by selecting the Project|String Table|Generate String File in Source Form command. This command causes PEG WindowBuilder to write a string file containing all of your String Id Names, your actual literal strings (Hex-Unicode encoded), an array of string pointers for each language, and a

function and macro for finding unique strings at run time. The string file is completely re-written each time the command is issued***, therefore you should never manually edit the string file***.

3.6 Source Code Generation

The end goal of running PEG WindowBuilder is to produce the C++ source code you will use to display your application screens. ***You will need to edit and add your own program logic to the source files produced by PEG WindowBuilder***. Most significantly you will need to add program logic to catch signals generated by your child controls. You may also need to make any number of other additions and changes to the source files produced by PEG WindowBuilder.

At the same time, you will want to be able to run PEG WindowBuilder again and again to modify your screens and update the source files without losing any of your hand-coded changes. This is not difficult to do as long as you understand how WindowBuilder updates your source files and follow a few simple rules.

When you instruct PEG WindowBuilder to produce/update the source files using the Project|Update|Source command, PEG WindowBuilder first looks to see if the source file already exists. If it does, PEG WindowBuilder enters "**Merge Mode**". In Merge Mode, PEG WindowBuilder is very careful not to lose any of your custom modifications. The rules are this: PEG WindowBuilder will find and re-write the section of the source file delimited by the start of the constructor and the comment line which reads:

```
/* WB End Construction */
```

To avoid losing your changes, never make any manual edits between the start of the class constructor and this comment delimiter.

PEG WindowBuilder also searches for the Message() member function, if present, and updates this function to contain any new PEG_SIGNAL cases not already present. PEG WindowBuilder will NOT remove case statements from your Message function, even if the control which generated a specific PEG_SIGNAL is no longer a child of the window. In short, deleting obsolete sections from your source files is your responsibility, in the interest of safety PEG WindowBuilder will not delete source lines from your Message function.

Any and all code outside of the class constructor and Message function is maintained without modification during the source code merge process. That is, any other editing you have done will be preserved entirely during the source file update process.

3.6.1 Pointer Name Control

You can control the type and name of the pointer (if any) used when each child object of the top-level window is created. Controlling how pointers are used is done by adjusting the basic properties, using the properties dialog, for each child control. There are four types of pointers used by PEG WindowBuilder during code generation: Member pointers, Automatic Named pointers, Automatic Temporary pointers, and Implicit pointers. We will describe each type below and describe how you can control the use of pointers in the generated source code.

Implicit Pointers

The most basic pointer type is the implicit pointer. An implicit pointer is used by PEG WindowBuilder when no references to an object are made after the object has been created and you have not chosen to create a member or automatic pointer. In this case PEG WindowBuilder does not need to keep the address of the newly created child in any variable, and therefore uses an in-line, “implicit” pointer to pass the child’s address to the Add() function. The following is an example of source code produced by PEG WindowBuilder which uses an implicit pointer:

```
Add(new PegPrompt(ChildRect, “Text”));
```

Note the return value from the new operator is not saved, but is passed directly to the Add() function. When no other pointer type is needed, this is the default pointer style used by PEG WindowBuilder.

Temporary Pointers

Next up in the PEG WindowBuilder source code generation process is the temporary pointer. This type of pointer is used by PEG WindowBuilder when reference to an object is required after it has been created, but you have not requested an automatic or member pointer be created. In this case, PEG WindowBuilder will create a temporary automatic pointer to hold the address of the child object instance. The temporary pointer is called “Automatic” because it is created on the execution stack, i.e. space for the pointer is allocated automatically by the compiler on the stack, and the space is destroyed when the function (in this case the class constructor) returns.

A common example of this might be a PegGroup container added to the top level window. During code generation, PEG WindowBuilder needs to maintain the address of the PegGroup instance while creating and adding child controls to the group. PEG WindowBuilder will default to using a temporary pointer for this purpose, which produces source code with the following appearance:

```
PegThing *pChild1;

pChild1 = new PegGroup(…); // keep temp pointer to object

pChild1->Add(…); // add second-generation children to object
pChild1->Add(…); // ditto

Add(pChild1); // add object to top-level window
```

PEG WindowBuilder will always use the generic names pChild***x*** for temporary automatic pointers. PEG WindowBuilder will reuse the temporary pointers for new objects if needed and available during code generation. In some cases, multiple temporary pointers are required simultaneously, in which case PEG WindowBuilder will create and use as many temporary object pointers as are needed.

Automatic Named Pointers

Similar to automatic temporary pointers, Automatic Named pointers are created on the execution stack and only exist during the class constructor. Named pointers are created by typing a name into the “Pointer Name” field in the object properties dialog basic properties page and **unchecking** the “Member Pointer” box. Note the name must be a valid C++ variable name or your compiler will flag an error when you compile the generated module (no name checking is done by PEG WindowBuilder!).

Automatic Named pointers are very handy to you, the developer, when you want to modify the object created by PEG WindowBuilder in ways that are not supported by the PEG WindowBuilder properties pages. An Automatic Named pointer is used only for the object in question, and more importantly is still valid and available to you at the end of the class constructor (i.e. after the ***“/* WB End Construction */”*** marker). This allows you to further modify an object by calling class member functions via the named pointer prior to returning from the class constructor.

Named pointers also help to improve the syntax and eliminate casting when PEG WindowBuilder must call member functions of a class. For example, if

PEG WindowBuilder must call the “SetFont” function for a PegPrompt, it will cast a temporary automatic pointer as follows:

```
((PegPrompt *) pChild1)->SetFont(…);
```

If an automatic named pointer is used, it will be a pointer to the desired type and no casting is required:

```
PegPrompt *MyPrompt;

MyPrompt = new PegPrompt(…)
MyPrompt->SetFont(…);
```

This can greatly improve the appearance and readability of the constructor source code produced by PEG WindowBuilder.

Member Pointers

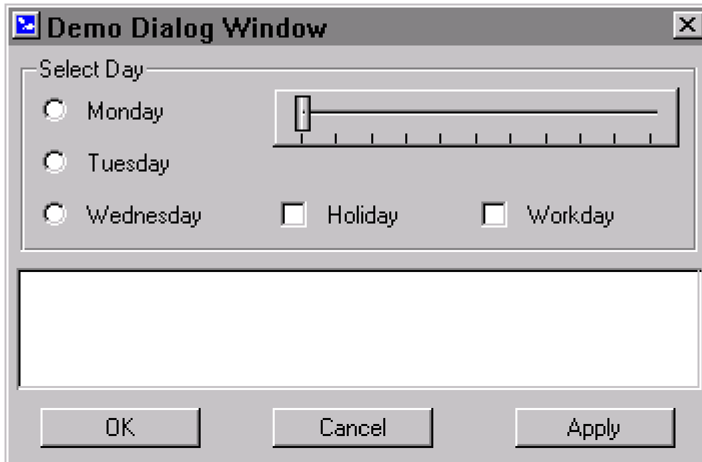
The final pointer option is the member pointer. A member pointer is a pointer to a child object which is maintained as a member variable of the parent window class. This pointer is initialized in the class constructor and used at all times to reference the child object. You can instruct PEG WindowBuilder to create a member pointer for a child object by checking on the “Member Pointer” checkbox in the properties dialog and typing a name in the “Pointer Name” field. Note the name must be a valid C++ variable name or your compiler will flag an error when you compile the generated module (no name checking is done by PEG WindowBuilder!).

3.7 Example 1: Creating a simple PegDialog window

In this example, we will walk step-by-step through the procedure required to create a new window builder project, create a new source module, and create a simple PegDialog derived window. This example takes about 15 minutes to complete.

The Example Dialog:

These instructions will take you systematically through the process of creating the simple dialog window shown below. In the following instructions, we will call this the ‘reference dialog’:

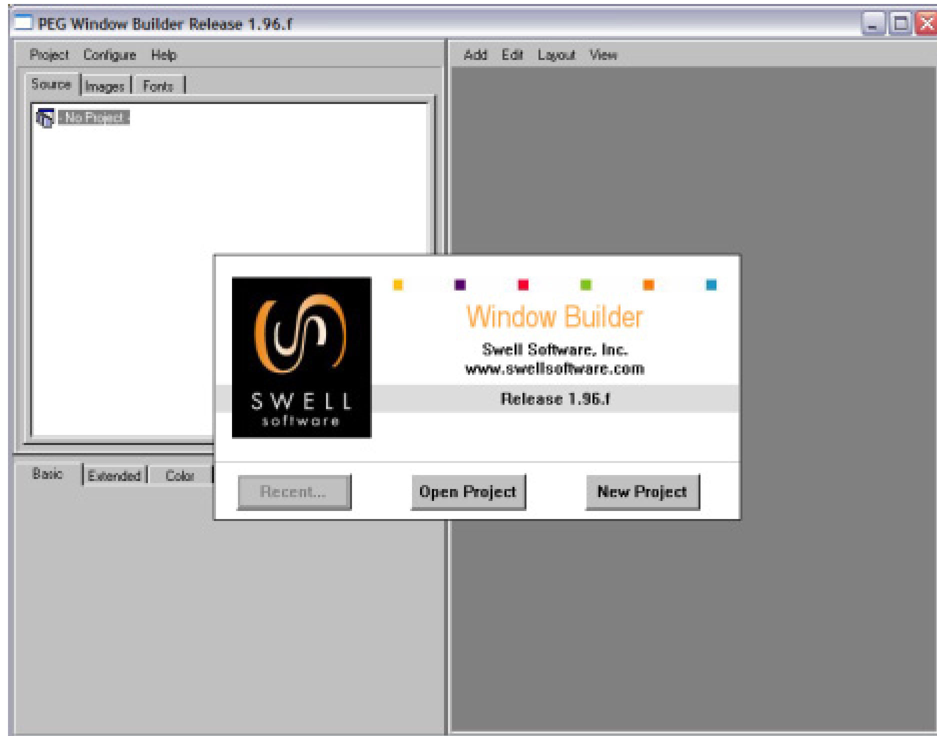


You may find it helpful to refer to the appearance of this dialog as you follow the instructions below.

3.7.1 Creating and Configuring a Project:

Under MS Windows 95/98/NT/2000, start the PEG WindowBuilder executable program, **pwinbld.exe.** You should see the screen shown here

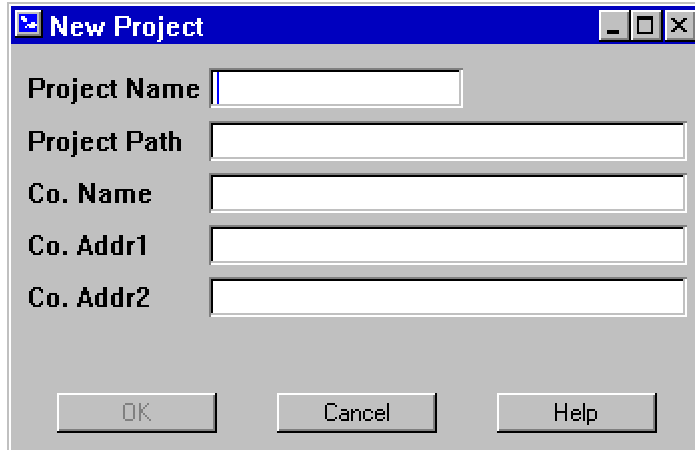
:



This is the PEG WindowBuilder startup screen. This screen allows you to quickly resume work on a previous project, or begin a new project. For this example, we want to begin a new project, so you should select the **New Project** button.

Step 1- Configure Project and Directories

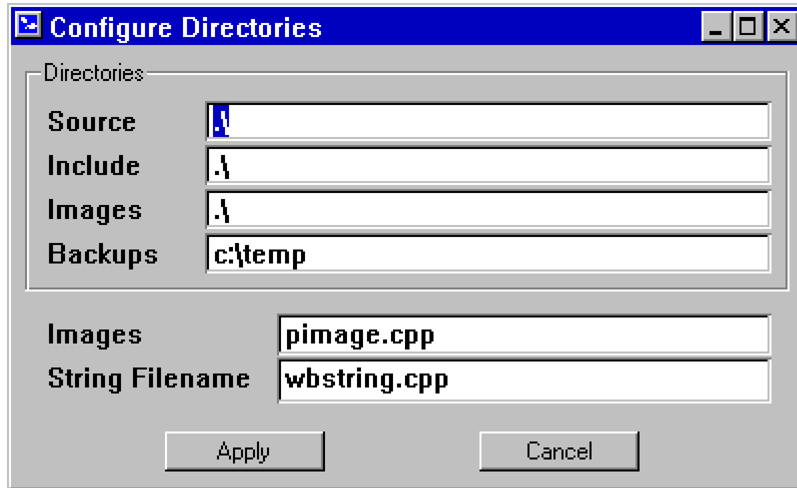
Whenever you begin a new project, PEG WindowBuilder asks you to enter some basic project information such as the project name and where to keep the project file on your computer. PEG WindowBuilder presents the following dialog window to allow you to enter this information:



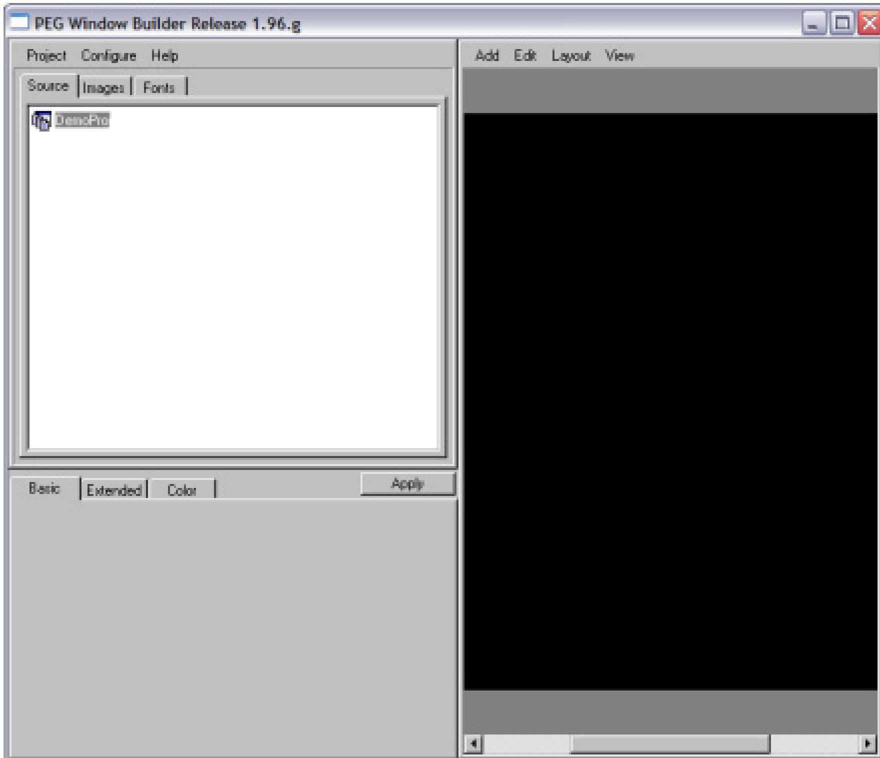
In the project name field of this dialog, type “**DemoProj**”. Your project file will be saved to this name. In the project path field, type any valid drive and directory name. If the directory does not exist, PEG WindowBuilder will create it when you save your project.

You can also enter your company name and address, although this is not required. If you do enter this information, PEG WindowBuilder will include copyright notifications in the header area of the generated source files.

After you have entered in the required information, select the “OK” button. You are now presented with the following screen:



This dialog allows you to tell PEG WindowBuilder where you would like to save the PEG WindowBuilder output files. For this demo, you only need to enter in valid path names for the **Source** and **Include** directories. You can leave these at the default settings, or enter in alternate directory names where you would like to save the source code produced by PEG WindowBuilder. It is acceptable to enter the same directory name for both source and include files. After you have entered the Source and Include directory names, select the “Apply” button. You will now see the screen on the following page:



This is the default appearance of PEG WindowBuilder. PEG WindowBuilder initially contains three main windows. These are the **Project** window (upper left), the **Target** window (right) and the **Preview** window (lower left).

The project window maintains and displays information about all of the source files, images, and fonts which are part of your PEG WindowBuilder project. The Target window provides an exact representation of your target system display screen. The Preview window allows you to view the images (i.e. bitmaps) and fonts you have added to your project.

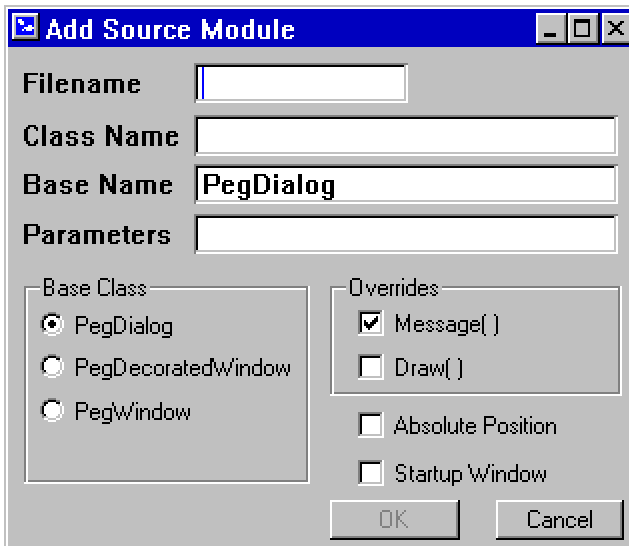
You should now see that the Project window displays the name of the project, i.e. “DemoProj” as the top node in the project source tree.

When a new project is created, the target window begins to display the target screen. The default target screen resolution is 640x480 pixels and uses 256 colors. This can be modified to any supported color depth and

resolution using the Configure|Target command. For this example, you should leave the target screen configuration at the default settings.

Step 2- Add a new Module

You are now ready to create a new module. Each PEG WindowBuilder module contains a unique class declaration and class implementation. Select the **Project|Add Module** menu command. You will now see a dialog asking you to enter information about the new class to be created, shown here:



In the filename field, type "**DemoDlg**". This is the name that will be assigned to the source and header files produced for this class. In the Class Name field, type "**DemoDialog**". This name will be assigned to the generated class. In the Parameters field, type "**int iCount**". Any information typed into the parameters field is passed directly to the dialog constructor. While this demo will not actually use the incoming parameter, it is useful to see how this affects the generated source code.

Click on the Startup Window checkbox to turn it on. This will cause PEG WindowBuilder to generate a default PegAppInitialize function for us which will display our example window. Leave the remaining dialog fields at their default values and select "OK".

Click on the DemoDialog icon, and the target window now displays the default dialog window which has been created.

3.7.2 Editing the Module:

Step 3- Modifying position and size.

You can select the default dialog by left clicking with the mouse anywhere in the dialog window. When the dialog is selected, a dark box is drawn around the dialog window to indicate it has been selected. You can now use the left-mouse button 'click-and-drag' operation to move the dialog window, and you can use the arrow keys on your keyboard to move the dialog window to any position on the target screen.

If you position the mouse pointer over the dark border around the dialog, the mouse pointer will change shape to indicate you can also re-size the dialog. You should experiment with moving and resizing the dialog until you are familiar with these operations.

Resize the dialog window under the Width field at the bottom of the screen which is approximately equal to "354", and the height is roughly "232". You don't have to be exact, but these are the approximate dimensions of the reference dialog window we are creating in this example.

Step 4- Modifying Properties

You can also set the position, size, and other properties of the dialog window by selecting the dialog and using the properties panel in the lower-left portion of the window. If the properties panel is not visible, make sure that the "Source" tab is selected in the project window.

The first properties page is called the "Basic" properties. These properties are always available, no matter what type of PEG object you are working with. In this case we can leave the Basic properties at the current settings.

Now select the "Extended" tab. This page of properties allows you to adjust parameters that are specific to the dialog window. In the Title field, type "**Demo Dialog Window".** This assigns the dialog window title. Now select the "Apply" button to apply your changes.

Step 5- Add a PegGroup to the Dialog

Make sure the dialog window is selected, and then select the menu command **Add|Container|PegGroup.** This will add a new PegGroup control to the dialog window***. The Add menu command always adds the selected object type to the previously selected parent.*** In this case, the parent is the dialog window and the new object type is a PegGroup control.

Use the mouse and arrow keys to size the group control so it is similar in position and size to the reference example. Edit the group properties by going to the Properties panel in the lower-left corner, and on the Extended properties page enter "**Select Day**" in the "Initial Text" field. This assigns the text value which is displayed as the group title. Select "Apply" on the properties dialog and you will see your changes take effect.

Step 6- Add Radio Buttons to the Group

Once you have the PegGroup in position, insure you select it by left-clicking inside the group with the mouse. Now select the **Add|Button| PegRadioButton** command. Following this, a new PegRadioButton is added to the center of the PegGroup. This is the general operation of the Add command, in that the selected type of object is created with a default size and positioned at the center of the object's parent area.

Use the arrow keys to move the radio button to the upper left corner of the group box, and then edit the radio button properties. On the Basic properties page, enter "**IDRB_MONDAY**" in the ID field. On the extended properties page, enter "**Monday**" in the Text field, and when you are done select "Apply". The ID value is the value you will use to identify the radio button during program operation. This value is saved in a list of enumerated control IDs in the generated class header file.

Repeat the above procedure to add the two additional radio buttons. Make sure you select the PegGroup parent object before adding each radio button, to insure that the radio buttons are children of the group object. For these buttons, assign the first the ID value "**IDRB_TUESDAY**" and the Text "**Tuesday**". For the last radio button, assign the ID value "**IDRB_WEDNESDAY**" and the text "**Wednesday**".

You can use the mouse and array keys to position the radio buttons in the approximate order and position you want them to be in. You don't have to be exact, we will use the Layout commands to insure that the radio buttons are perfectly aligned.

Step 7- Using Layout commands.

To insure that the radio buttons are equally aligned, we can use the Layout commands. The layout commands effect collections or groups of objects. In this case, we want to select all three radio buttons before using layout command.

To select the three radio buttons, first select the top radio button with the text value Monday by left-clicking on that radio button. Now hold down the <ctrl> key and left click on the “Tuesday” and “Wednesday” radio buttons in turn. You will see the selection box grows to enclose all three radio buttons.

Now we want to use the **Layout|Align|Left** command to align the left edge of the radio buttons. After selecting this command, you should see the radio buttons are all exactly aligned at the left border.

Note while you have multiple-objects selected, you can use the mouse and arrow keys to move all of the objects as a group. Use the arrow keys now to slide the three radio buttons into a position that “looks right”.

Step 8- Add remaining children to Group.

You can add the two checkbox objects to the group by first selecting the Group box, and then selecting the **Add|Button|PegCheckBox** command. Position the checkboxes using the same methods described above. Assign the first check box the ID “**IDCB_HOLIDAY**” and the Text “**Holiday**”. Assign the second check box the ID “**IDCB_WORKDAY**” and the Text “**Workday**”.

Again select the group box, and select the **Add|Slider/Scroll|PegSlider** command. This adds a PegSlider control to the group box. Use the mouse and arrow keys to position and size the slider control as shown in the reference diagram. You do not need to assign any additional properties to the PegSlider control.

Step 9- Add PegTextBox

Click on an unused portion of the dialog window to select the window. Be sure the dark border encloses the entire dialog window. A common mistake is to click inside of the group box, in which case the group box is selected rather than the dialog window. Now select the **Add|Text|PegTextBox** command. A default size textbox is positioned at the center of the dialog window. You will need to reduce the height of the textbox using the mouse or properties dialog, and move the text box so it is underneath the group. You can also use the properties dialog to enter an initial text value, such as “**Hello World**”.

Step 10- Add TextButtons

Repeating the above procedures, click on an unused portion of the dialog window, and then select the **Add|Button|PegTextButton** command to add a new button to the dialog window. The button will again appear at the center of the dialog, and you will need to use the mouse or arrow keys to move the button into position.

Create the three buttons at the bottom of the dialog one at a time, repeating the above process. Use the edit properties command to assign the Text values “OK”, “Cancel”, and “Apply” to each button. Likewise, assign the ID values “IDB_OK”, “IDB_CANCEL”, and “IDB_APPLY” to each button, respectively.

You can use the **Layout|Align|Top** command to insure the buttons are vertically aligned, and move them as a group until they are centered on the dialog window.

You are now done creating the dialog window!!

3.7.3 Saving Your Work:

Step 11- Save the project

At this point you should select the **Project|Save** command to save your project. This will create the file “DemoDlg.wbp” in the directory you selected in the Configure Directories dialog. Once you have saved your project, you can later open it at any time and modify this dialog or add any number of additional modules to the project.

Step 12- Generate Source Code

Make sure the module “DemoDialog” is selected in the project tree (it should be highlighted). If it is not, left-click with the mouse in the project source tree or use the arrow keys to select the DemoDialog module. Now select the **Project|Generate Source|Current Source Module** command to create the C++ source files corresponding to this module. After the source code has been generated, PEG WindowBuilder should inform you the source or header file has been updated.

Step 13- Close WindowBuilder

Select the **Project|Exit** command to exit the PEG WindowBuilder application.

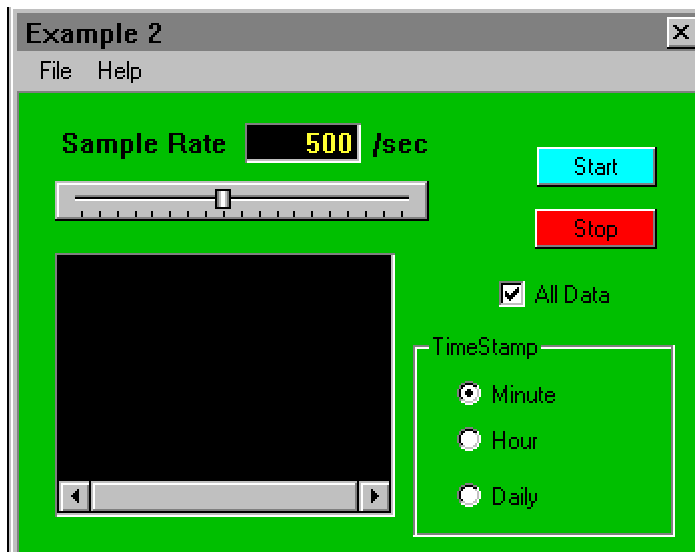
3.7.4 Examining the Source Code:

You can now open the source and header files produced by PEG WindowBuilder with your favorite text editor. The generated .cpp file contains the exact C++ source code required to construct the example dialog window at run time, and the generated .hpp header file contains the class declaration and ID enumeration necessary to complete the class description. The source code file also contains a skeleton of the message processing function you will edit to make your dialog do real work.

3.8 Example 2: An advanced PegDialog window

For this example, we will instruct PEG WindowBuilder to create a new PegDecoratedWindow derived window class, and to generate an overridden message handling function. We will also use the 'Maintain Pointer' and 'Send Signals' options to make the dialog do useful work. When you are done creating the window, you can actually compile the source code and run the application! Be forewarned, this example takes about 60 minutes to complete.

The final appearance of the new window is shown below:



In order to make the example more interesting, assume this window will display a continuous time chart of some analog data value. For this reason, we have provided the user with a PegSlider control to adjust the 'Sample

Rate', Start and Stop buttons to start and stop the sampling process, and a range of timestamp values that indicate how often the recorded data will be saved to a permanent storage media.

We will also add a menu bar to the window. While this is not necessary for this example, it gives us a good chance to demonstrate how you can create and customize menus with PEG WindowBuilder.

Step 1: Create a Project

Start PEG WindowBuilder and create a new project as in example 1. Name this project 'Example2'. Use the **Configure|Directories** command to set both the source and include output directories to …\peg\wb\example2. This will enable you to build the resulting application using the provided project file.

Step 2: Add a Module

Create a new module using the **Project|Add Module** command. Enter **exam2** as the file name, and enter **ExampleTwo** as the class name. It is normally up to you to decide on the file and class names; however, we have provided a project file and a version of the PegAppInitialize function which will allow you to build and execute the window after you are done, so in this case it is best to stick with the suggested names.

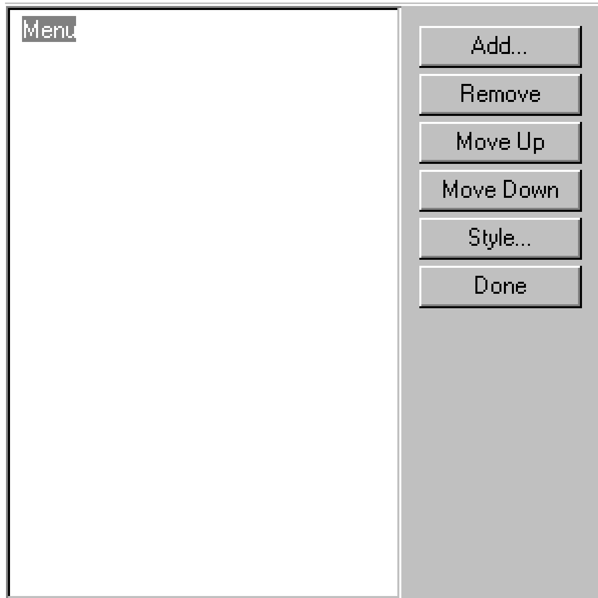
Before you click on the "OK" button, select the "PegDecoratedWin" button in the Base Class group. This means your new window will be derived from PegDecoratedWindow, rather than the default PegDialog. Leave the other selections at their default settings.

Step 3: Customize the Window

Use the properties dialog to modify the appearance of the window until it resembles the reference example. You will need to select the 'Color' tab, and set the PCI_NORMAL color. You will also need to select the Title Style button on the Extended properties page, and turn off the "System Button" and "Min/Max Button". You should also type "Example Two" in the window title field. On the basic properties page, set the window Height to 280 pixels, and the window Width to 350 pixels. Once you have done this, close the properties dialog by selecting the "OK" button. You should now see the window has the indicated fill color, and the size and title should match the reference example.

Step 4: Add the Menu Bar

Again enter the window properties dialog. On the Extended properties page, select the “Menu Bar” checkbox. This adds a blank menu bar to the window. Select the “Edit” button next to the Menu Bar check box, and you will see the dialog window below:



This is the PEG WindowBuilder menu editor. The left-hand panel is a PegTreeView control, which will show you all of your menu commands in a convenient tree format. The right-hand panel contains several buttons you will use to edit the menu bar.

For this example, we are going to create two top-level menu options, called “File” and “Help”. Click on the “MenuBar” node of the tree (something always has to be selected before the menu editing buttons are operational), and then select the “Add” button. You are presented with a small window which allows you to enter the menu “Caption” (the displayed menu text), the menu item ID, and the menu item style.

Type “File” in the Caption field, leave the remaining fields at their default value, and select “OK”. You now see the file menu item displayed in the tree!

Ensure the “MenuBar” tree node is still selected, and repeat the above procedure to create the “Help” item on the menu bar.

Now we can create any number of submenus. Select the “File” node in the tree, and add the following items:

Caption	CHILD ID	Style
Open	IDB_OPENFILE	AF_ENABLED
Close	IDB_CLOSEFILE	AF_ENABLED
		BF_SEPARATOR
Exit	IDB_CLOSE	AF_ENABLED

The style settings are exactly the same as setting the different Style flags when defining a PegMenuDescription. In fact, PEG WindowBuilder will use the information you enter to generate the required PegMenuDescription in source code format for you.

For this example we will not be using the MenuBar commands, so feel free to experiment with the Menu Editor and modify the MenuBar settings. Repeat the above procedures to add “About”, “Search”, and “View” commands to the top level “Help” button.

When you are done, select the “Done” button, and close the properties dialog by selecting “OK”. Your menu has now been added to the decorated window. You won’t be able to actually view the sub-menus from within PEG WindowBuilder, since selecting any non-client window object is intercepted by PEG WindowBuilder to select the parent window. However, you can re-open the properties dialog and edit the menu at any time.

Step 5: Add the Child Controls

We are not going to detail each of the child controls, as by this time you are probably becoming quite adept at adding new controls using the PEG WindowBuilder menu commands. There are, however, a few things we should point out to help you create a working window which looks like the reference example.

The black window in the lower-left corner is a PegWindow, added with **Add|Window|PegWindow.** We have modified the Frame style to be ‘Recessed’, changed the PCI_NORMAL color to black, and set the H-Scroll

mode to “Always”, which means the window will always display a horizontal scroll bar.

The fields “Sample Rate” and “/sec” are PegPrompt objects.

The field which displays the Sample Rate, and shows the value “500”, is also a PegPrompt object. This object is important for this example so we will detail its configuration more completely.

In order to obtain the appearance shown, you must first disable the “Transparent” style in the prompt Extended properties. When the Transparent style is enabled, the prompt always assumes the background color of its parent, so setting the prompt color has no effect. We have also set the PCI_NORMAL color to black, and the PCI_NTEXT color to green. Finally, the prompt Frame is set to “Recessed”, and the justification (also on the extended properties page) is set to “Right”. Also, select the “Copy Text” checkbox on the extended properties page. This instructs the PegPrompt object to make a copy of the text it displays, which is required when the text value assigned to the prompt is created dynamically on the stack. We will explain this further later in this example.

Set the prompt ID to IDP_RATE, and set the prompt Initial Text to “500”. Click on the “Member Pointer” checkbox in the prompt Basic properties page, and type in the pointer name “mpRatePrompt”. This instructs PEG WindowBuilder to define and use a pointer to this PegPrompt object which will become a member variable of the parent window. We will use this pointer to modify the prompt value as the slider control is adjusted. While we could also find a pointer to the prompt at run-time using the Find() function, in this case it is more convenient to simply tell PEG WindowBuilder to create and keep a pointer to the prompt.

The slider control is created with a Recessed frame, a minimum value of “100”, a maximum value of “1000”, and an initial value of “500”. Set the Tick Interval to 50, which means that a tick mark will be drawn at each increment of 50 in the slider range. Use the slider properties dialog to enter these values. Also, insure that the “Catch Signals” checkbox is selected on the slider control Basic properties page. This tells PEG WindowBuilder we want to receive notification messages from the slider control (in this case PSF_SLIDER_CHANGE signals), and PEG WindowBuilder will create the matching case statements in the window message handling function. Set the slider ID to “IDSL_RATE”.

The Start, Stop, and other remaining child controls are not required for this example. You do not need to assign ID values or select the “Catch Signals” checkbox for any of these controls. Simply create them and place them on the window to achieve the appearance shown above.

Step 6: Save Your Work!

Use the **Project|Save** command to save your project file. You don’t want to lose all of your effort so far do to a power failure or other malady!

Step 7: Generate Source Code

Now select the **Project|Generate Source|Current Source Module** command to generate the source code for your window.

Step 8: Add Message Handling

Open the generated source file in your favorite editor. You will see PEG WindowBuilder has generated a constructor for the ExampleTwo window, and also the message handling function. Your message handling function should appear very similar to the function below:

```
PEGINT ExampleTwo::Message(const PegMessage &Mesg)
{
    switch (Mesg.wType)
    {
    case PEG_SIGNAL(IDB_HELPINDEX, PSF_CHECK_ON):
        // Enter your code here:
        break;

    case PEG_SIGNAL(IDB_HELPINDEX, PSF_CHECK_OFF):
        // Enter your code here:
        break;

    case PEG_SIGNAL(IDB_ABOUT, PSF_CLICKED):
        // Enter your code here:
        break;

    case PEG_SIGNAL(IDB_CLOSE, PSF_CLICKED):
        // Enter your code here:
        break;

    case PEG_SIGNAL(IDB_FILESAVE, PSF_CLICKED):
        // Enter your code here:
        break;

    case PEG_SIGNAL(IDB_OPENFILE, PSF_CLICKED):
        // Enter your code here:
        break;

    case PEG_SIGNAL(IDSL_RATE, PSF_SLIDER_CHANGE):
```

```
        // Enter your code here:
        break;

    default:
        return PegDecoratedWindow::Message(Mesg);
    }
    return 0;
}
```

Note message case statements have been generated for all of the MenuBar commands, and also for the window child controls for which the "Send Signals" option was selected. In this case, verify you have the case statement "case PEG_SIGNAL(IDSL_RATE, PSF_SLIDER_CHANGE):" in your message function. We are going to add the source code here required to make the prompt object display the current slider value. If you do not see this case statement, you should check the properties of the slider control with PEG WindowBuilder, and insure the "Catch Signals" checkbox is checked.

Modify the case statement as shown below:

```
case PEG_SIGNAL(IDSL_SLIDER, PSF_SLIDER_CHANGE):
    {
        PEGCHAR cTemp[40];
        ltoa(Mesg.lData, cTemp, 10);
        mpRatePrompt->DataSet(cTemp);
        mpRatePrompt->Draw();
    }
    break;
```

The above code will convert the slider value, which is passed in the message lData field, into an ASCII string for display. It then assigns this string value to the prompt, and re-displays the prompt. Note the pointer ***mpRatePrompt*** has been created for us by PEG WindowBuilder, and is defined in the exam2 header file as a private member variable of the Example2 class.

A less obvious note should be made. If you remember we instructed you to select the "Copy Text" checkbox in the range prompt Extended properties page. Normally, PEG objects do not copy text strings in order to reduce memory usage. However, in this case we are creating a new string on the execution stack, via the automatic "cTemp" variable. In this case, we want the prompt object to copy the assigned text, since the cTemp variable will not exist once we return from the Message function.

Step 9: Build and run the program

We have provided a version of the startup function PegAppInitialize which creates and displays the ExampleTwo window in the \peg\wb\example2 directory. This file is called startup.cpp. In order to build and run the program, you will need to do the following (using MS VC++)

a) Create a new workspace.

b) Add the project file \peg\build\win32\ms60\peg.dsp to your workspace. The project builds the PEG library.

c) Add a new project to the same workspace. This project will build the example application. You can name the project anything you like. After you have added the project, add the files “startup.cpp” and “exam2.cpp” to your project.

d) Make sure your project dependencies list “peg.lib”, as this will build and link the PEG library with your application files.

e) Build and run the program!

As the program executes, you should be able to adjust the slider control and see the value displayed in the prompt object. You have used PEG WindowBuilder and a little hand coding to create a complex window! We hope you have enjoyed working through this example program.

3.9 Customizing PEG WindowBuilder

It is possible to customize PEG WindowBuilder to include and utilize your own custom gadgets, controls, and windows. This allows you to design your windows and dialogs using modified objects which have your own look-and-feel. In this section we will describe how to build a custom WindowBuilder executable program that contains your custom gadgets and windows.

Once you have built the custom executable program as described below, PEG WindowBuilder will have compete access to your custom objects. Your custom objects will be created by PEG WindowBuilder when requested, and will draw themselves and handle messages just like standard PEG library objects.

An example project and source files for making a custom form of the WindowBuilder program is provided in the \peg\wb\custom\ directory.

Requirements and Limitations

The ability to customize PEG WindowBuilder can be a very powerful feature, but it does require some effort on your part and there are a few minor requirements and limitations. These requirements and limitations include:

- Each custom object must be derived from a class in the PEG library, and must not use multiple-inheritance.

- Each custom object must have a constructor identical in parameters to the base class common constructor, meaning your custom object must accept the same parameters as the base class constructor used by PEG WindowBuilder when creating that object type. This is always the first constructor listed in the reference manual. Your custom object may have additional constructors, but they will not be utilized by PEG WindowBuilder.

- If your custom object requires configuration parameters beyond those provided in the class constructor, this configuration should be done via custom APIs that you define for your custom object. This will insure the source code produced by PEG WindowBuilder will be fully compatible and can be continuously updated without losing any information.

- All strings must be Unicode encoded, and use two-byte character encoding. This requirement is almost completely transparent to your custom object code provided that you use the PEGCHAR data type for all character data. Even if you will not be supporting Unicode on your target, this requirement does not impose any special burden on how you code your custom object.

- Your custom object must be able to draw itself in TrueColor (24-bit-per-pixel) mode in addition to your target color mode. The reasons for this are many, but in brief PEG WindowBuilder runs in TrueColor mode with a custom TrueColor screen driver, and your object must pass 24-bit color values when drawing itself in the PEG WindowBuilder environment. Since most target systems which utilize PEG are not TrueColor systems, this may require you define two color sets or two “Draw()” functions. The first version will be used when running with PEG WindowBuilder, and the second version will be used when running on your target. You will use this pre-processor construct:

```
#if defined(WINDOW_BUILDER)
```

```
// Draw in TrueColor mode

#else

// Draw in my target color mode

#endif
```

in the areas of your custom object code which deal with color values. While this might sound ominous, in reality it is simple to do and we will illustrate how to accomplish this in the example provided. Also, this requirement does NOT apply to bitmaps your custom object may draw. PEG WindowBuilder will properly display bitmaps of any color depth, so if your custom object draws an image or images using the PegScreen::Bitmap function, these images need only be provided in the appropriate color depth for your target system.

Finally, to customize PEG WindowBuilder you will need a compiler and linker capable of producing a 32-bit Win32 application program, and compatible with a 32-bit library produced by the Microsoft compiler and librarian. The examples below use the Microsoft VC++ 6.0 compiler, but any Windows compatible compiler can be used.

3.9.1 Step-by-Step

To get started, we will describe each step in creating a custom WindowBuilder executable file and how it works. Once the process is understood, we will work through a complete example which adds two custom objects to the PEG WindowBuilder program.

Step 1: Define your class

The first thing to do is to write your custom class as you normally would. This means you will define your own class, and it will be derived from a PEG library class. You should try to pick the PEG base class which is closest in appearance and functionality to what you desire in your custom object. Most frequently, you will override the Draw() function to customize the appearance of your class, but this is not required. You can override as many or as few of the base class virtual functions as you desire to arrive at your custom object. You can also add your own API or public functions for your class. PEG WindowBuilder does not need to know about these custom API functions, they will only be accessible to you when you write your application software.

When defining and implementing your class, remember you will need to provide a constructor identical to the base class constructor. For example, suppose your custom object is derived from PegPrompt, and is named "MyCustomPrompt". From the PEG reference manual, we see the first PegPrompt constructor looks like this:

```
PegPrompt(const PegRect &Rect, const PEGCHAR *Text, PEGUSHORT wId = 0,
    PEGUSHORT wStyle = FF_NONE|TJ_LEFT|AF_TRANSPARENT)
```

From this, we know the constructor for MyCustomPrompt must look like this:

```
MyCustomPrompt(const PegRect &Rect, const PEGCHAR *Text, PEGUSHORT wId = 0,
    PEGUSHORT wStyle = FF_NONE|TJ_LEFT|AF_TRANSPARENT)
```

You can add additional constructors if you like, but only this constructor is required for compatibility with PEG WindowBuilder.

Repeat step 1 for each custom class you want to add to PEG WindowBuilder. There is no limit to the number of custom classes which may be defined.

Step 2: Define your WindowBuilder Resources

PEG WindowBuilder defines a structure containing important information about each of your custom classes. This structure is type defined like this:

```
typedef struct {
    const PEGCHAR *pResName;
    PEGUSHORT wType;
    PEGUSHORT wBaseType;
    PEGUSHORT wMenuCommand;
    PEGUSHORT wFrameStyles;
    PEGUBYTE uColorSet;
    PEGUBYTE uReserved;
    PEGBOOL      bSizeable;
    PEGBOOL      bAddMenu;
    PEGBOOL      bModuleBase;
    PEGBOOL (*Reserved1)(void *, void *, void *);
    PEGBOOL (*Reserved2)(void *, void *);
    PegThing  *(*DefConstruct)(PegRect &, PEGBOOL);
} WB_RESOURCE;
```

pResName is a pointer to 2-byte encoded string, the name of your custom class.

wType is your custom class type. All custom classes used in PEG WindowBuilder must have a unique type.

wBaseType is the PEG library object type from which this class is derived.

wMenuCommand is filled in by PEG WindowBuilder, and should be set to zero (0).

wFrameStyles indicates which frame styles this object supports. The most common settings are 0 (none), and ALL_FRAME_TYPES.

uColorSet indicates which, if any, color values can be assigned by PEG WindowBuilder. Definitions are provided in the wbres.hpp file which can be "or'd" together to build the value of uColorSet.

uReserved should be set to zero (0).

bAddMenu indicates whether or not this custom object should appear on the PEG WindowBuilder Add menu as a custom child object type.

bModuleBase indicates whether or not this custom object can be used as a base class for a module. Both bAddMenu and bModuleBase can be TRUE, they are not exclusive.

Reserved1 and ***Reserved2*** are function pointers for future use, and should be set to NULL.

DefConstruct is the address of the function to call to construct this object.

You will create an array of WB_RESOURCE structures, each entry in the array will define one of your custom objects. This array has already been created for you in the example provided, but you will of course want to add to or modify the array provided to insert your custom objects.

Step 4: Build the pwinbld.exe Program

Now that everything is in place, you are ready to build the WindowBuilder program file. A Microsoft project workspace is provided in the \peg\utils\wb\ custom directory to serve as an example, but we can outline the important compiler/linker switches here in case you are using another compatible compiler.

The project must be a “Win32 Executable” project.

Under compiler settings, structure packing should be set to “8-bytes”, “Exception Handling” should be disabled, and “RTTI” (RunTime Type Identification) should be disabled. You can perform a debug-mode build while debugging your custom software, but you will not have debugging information for the WindowBuilder library itself. Once you are satisfied that everything is working, we recommend that you do a “Release Mode” build, meaning that you should NOT include any debugging information in your custom wbcustom.exe program.

While building the executable file, your include directory path should be set to \peg\utils\wb\custom\include, and should NOT include your standard PEG library include path, \peg\include. ***This is important!*** To insure all of the library configuration flags and build settings are identical to those used when building the PEG WindowBuilder library, a complete snapshot of the properly configured PEG header files is included in the \peg\utils\wb\custom\include directory. Use this header directory only for the special case of building the WindowBuilder executable. Never modify these header files, and remember to go back to the standard header files in the \peg\include directory for all other builds either for an emulation environment or for your target environment.

That’s it! Next we will work through a functional example, from which you should find it easy to add your own custom objects.

3.9.2 The SwellButton and SwellScale

An example is provided to put the above description into practice. The example files are found in the directory \peg\utils\wb\custom. We suggest you open each of these files in your favorite editor as we describe them.

prdbutn.cpp
prdbutn.hpp

These files contain a custom button class, derived from PegBitmapButton. This class draws a fancy border using many colors and bitmaps.

You will notice the first function in this file is named *CreateRndTextButton.* This function is only included in the file if the definition WINDOW_BUILDER is defined. This function is called by PEG WindowBuilder when the user wants to add a RndTextButton to some other object using normal PEG

WindowBuilder operations. There are two parameters to this function: **PegRect Size and PEGBOOL bFirstInstance**.

PEGBOOL bFirstInstance

The boolean **bFirstInstance** parameter indicates if the object is being newly created. For example, when the PEG WindowBuilder user Invokes the Add command, this function will be called with bFirstInstance == TRUE. This tells the function to create an object of default size. The default size is defined by you, but should fit within the **Size** parameter. The **Size** parameter in this case is the mClient region of the parent object. You can, if you like, use this **Size** parameter to create an object which is proportional to the parent object size.

If the **bFirstInstance** parameter is FALSE, this means the button size is known by PEG WindowBuilder. The function should create an object exactly the size passed by PEG WindowBuilder in **Size**.

prdscale.cpp

prdscale.hpp

These files are similar, they define a custom scale indicator derived from PegLinearScale.

wbres.hpp

This file defines the WB_RESOURCE structure. This structure is used to pass information about your object properties to PEG WindowBuilder. An array of these structures is defined in the custom.cpp file. The WB_RESOURCE structure is defined below:

custom.cpp

This is the main module of the custom executable. This module contains the array of WB_RESOURCE structures, and exports the callback functions used by PEG WindowBuilder to create instances of your custom class. To add your own class(es), all you need to do is add entries to the WB_RESOURCE array, and declare your function names which will be called to construct your objects.

usertype.hpp

This file contains an enumerated list of custom object types. These types are very important, because PEG WindowBuilder uses this type number to identify each resource when loading and saving project files. The type

numbers must be unique, and cannot be changed after you have created a .wbp project file using your custom types.

3.9.3 Building wbcustom.exe

A MSVC++ project file and workspace are provided to build the wbcustom.exe program. You do not have to use the MSVC++ tools, but this is the toolset used for the example.

You will need to define your own classes as described above, and edit the custom.cpp and usertype.hpp files to add your class information. Follow the examples provided, and do not modify the custom.cpp file other than adding your own class names, constructor functions, and WB_RESOURCE structure definitions. You should not modify any of the C functions in this file.

Open the workspace "\peg\utils\wb\custom\custom.dsw" using the Microsoft MSVC++ IDE. You will see that this project contains the files described above, including the wblib.lib library. Also, make sure you use the "Tools|Options|Directories" menu command, and set your include file path to ***\peg\utils\wb\custom\include***. The path ***\peg\include*** should **not** be in your include path listing.

Build the wbcustom.exe program file. This is your own custom version of the WindowBuilder executable program, and it will have access to all of your newly created custom object types.

3.9.4 Additional Notes

WindowBuilder uses your custom object “wType” field to store your custom object information in your WindowBuilder project file. When WindowBuilder reads your project file and finds one of your custom object types, it uses the custom resource definition list to call the correct constructor for your custom object(s).

This can lead to difficulties if you ever modify your custom object wType definitions. To handle this, WindowBuilder calls the WBCustomVersion function at startup to see if the custom version matches the version saved in your project file. If not, WindowBuilder presents a dialog that allows you to provide translation information for your custom object types.

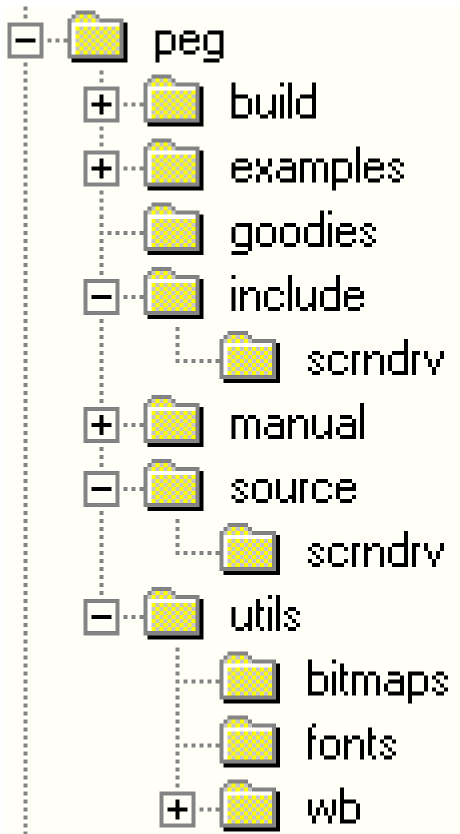
For example, suppose you create a custom WindowBuilder program using custom object types 0x80 and 0x81. You use WindowBuilder to create several modules and save your project. Now you decide to add a new

custom object type, and you want this new object to be type 0x80, bumping the previous custom objects to types 0x81 and 0x82. In this case, you should increment the #define WB_CUSTOM_VERSION before building the WindowBuilder executable. WHen WindowBuilder reads your old project file, it will recognize that the custom version has been modified, and ask you to type new wType values. In this case, you would tell WindowBuilder that type 0x80 is now 0x81, and type 0x81 is now 0x82. Once you save your project these new object type numbers will be stored and you will not have to enter these translations unless you again modify your custom object type number definitions.

APPENDIX A:

PEG DIRECTORY STRUCTURE

This section provides an overview of the directories created during PEG installation and the contents of each directory. After installing PEG, you will find the following directory tree, starting at the root \peg node:



build- Various make files for building the PEG library using different compilers for different targets. Microsoft, Borland, and several other compiler-specific build files are provided in this directory.

examples- Each folder under this directory contains a complete working example program. Microsoft developer studio project files are also provided for quickly building each of the example programs.

goodies- Miscellaneous gadgets used in the example programs. Not part of the core PEG library.

include- PEG library header files.

include\scrndrv- Target specific PegScreen implementations.

manual- Online documentation, starting with index.htm

source- Source files for the core PEG library.

source\scrndrv- Target specific PegScreen implementations.

utils\bitmaps- This directory contains the PegImageConvert executable program.

utils\fonts- This directory contains the PegFontCapture executable program.

utils\wb- This directory contains the WindowBuilder executable program and examples.