

PEG[®]

Portable Embedded GUI

PEG WindowBuilder Manual ***PEG Pro/PEG+ Library Release 2.x***

Rev. 4
May 2011



© Copyright 2007-2008, 2011
Swell Software, LLC. All rights reserved.

**Copyright 2007–2008, 2011
Swell Software, LLC. All rights reserved.**

© Copyright 2007–2008, 2011
Swell Software, LLC.
6501 William Cannon Drive West
Austin, TX 78735
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, LLC.

All rights reserved.

**PEG is a registered trademark of Swell Software, LLC, Reg. U.S.
Pat. & Tm. Off. C/PEG, PEG Pro, PEG+, and PEG Windowbuilder
are trademarks of Swell Software LLC. All other product or
service names are the property of their respective owners.**

FORWARD

We at Swell Software thank you for choosing PEG!

PEG is by far the most used, best supported, and most adaptable graphical interface software available. Our industry-leading real-time operating system support, hardware integration, and development tool compatibility allow complete flexibility as you and your team create next-generation products.

The PEG library and development tools have today been used to create several thousand unique products, and those products have shipped shipped many hundreds of millions of units. The applications utilizing the PEG software cover a broad spectrum including various consumer electronics, medical instrumentation, video games, military communications, aeronautics, office equipment, and even desktop applications.

We hope that your own efforts will be equally successful, and we encourage you to utilize our technical support if you run into any speed-bumps along the way.

In addition to the PEG software 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.

How are the manuals organized

Your documentation set includes four separate manuals:

- 1) The QuickStart Guide.
- 2) The Programming Manual.
- 3) This PEG WindowBuilder User's Manual.
- 4) The API Reference Manual.

Forward

The Quickstart Guide is a short tutorial enabling you to easily begin working with WindowBuilder and create and run your own application in one of our supported desktop environments.

This programming manual provides an ‘under the hood’ view of the PEG software library internals and introduces basic concepts that are needed to fully understand how PEG works. This is followed by descriptions of the fundamental PEG classes.

The WindowBuilder User’s Manual is a guide to the operation of our WindowBuilder WYSIWYG development tool and resource manager.

The API Reference Manual provides extensive information about the fundamental PEG classes. This manual details the Application Programming Interface (API) of the PEG graphics library.

Each of these manuals are provided in both printed and electronic (PDF) formats. The PDF format manuals are always the most recent manual updates, while for practical reason the printed manuals can sometimes be a few months out of date.

Whenever the electronic manuals are updated they are posted to the Swell Software www site. 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.

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 software.

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 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 primarily at custom 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. 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.

In addition to the class library, PEG provides tools for generating graphical fonts, processing, optimizing, and compressing graphical images, designing screens and child controls, creating custom colors, and maintaining multi-lingual string data.

What PEG is NOT

PEG is not an operating system. While PEG can run completely standalone, the library does not provide software for system boot-up, task switching, file system, or any of the other operating-system level functions your product may require.

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.

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

Forward

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.

CHAPTER 1

PEG WINDOWBUILDER™

1.1 Overview

WindowBuilder™ is a rapid prototyping and design tool used to quickly create your PEG windows and dialogs. PEG WindowBuilder is also instrumental in organizing and maintaining the **resources** (Images, Fonts, Colors, and Strings) you will use in your application program. PEG WindowBuilder is provided as a Win32 or X11 executable program, depending on your chosen development environment.

1.1.1 How this manual is organized

This manual first describes each screen area of the WindowBuilder program in general terms, and provides a quick overview of each of three ‘modes of operation.’ The idea is to give you a quick general overview before diving into the details of each command and operating mode. The middle sections describe each menu command and operation of WindowBuilder in detail. Finally, at the end of this manual we put everything into practice by using WindowBuilder to create a simple but complete graphical application.

1.1.2 WindowBuilder Project Files

All the work that you do while running PEG WindowBuilder is saved in an internal structure called your WindowBuilder **Project**. When saved to disk this data structure is written in XML format and becomes your WindowBuilder **Project File**. WindowBuilder project files have the extension ‘.wbx.’

PEG WindowBuilder™

The project file accurately maintains information about the source files, target system, images, strings, fonts, and so on 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.

1.1.3 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 that you can easily copy your WindowBuilder project files from one computer to another as long as you also copy all related font and image files and maintain the same subdirectory structure for your project (if any) in all cases.

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

1.1.4 Source Output Files

The goal of WindowBuilder is to produce C or C++ source files, depending on the version of the PEG library you are using (PEG is available in both ANSI C and C++ forms). These source files are intended to be ready to compile and run on your target system. All of the layout, property settings, images, fonts, etc. that you use while running PEG WindowBuilder will at some point be exported in the form of C/C++ source files.

For most .cpp source files, PEG WindowBuilder also creates a corresponding header file. These header files contain class

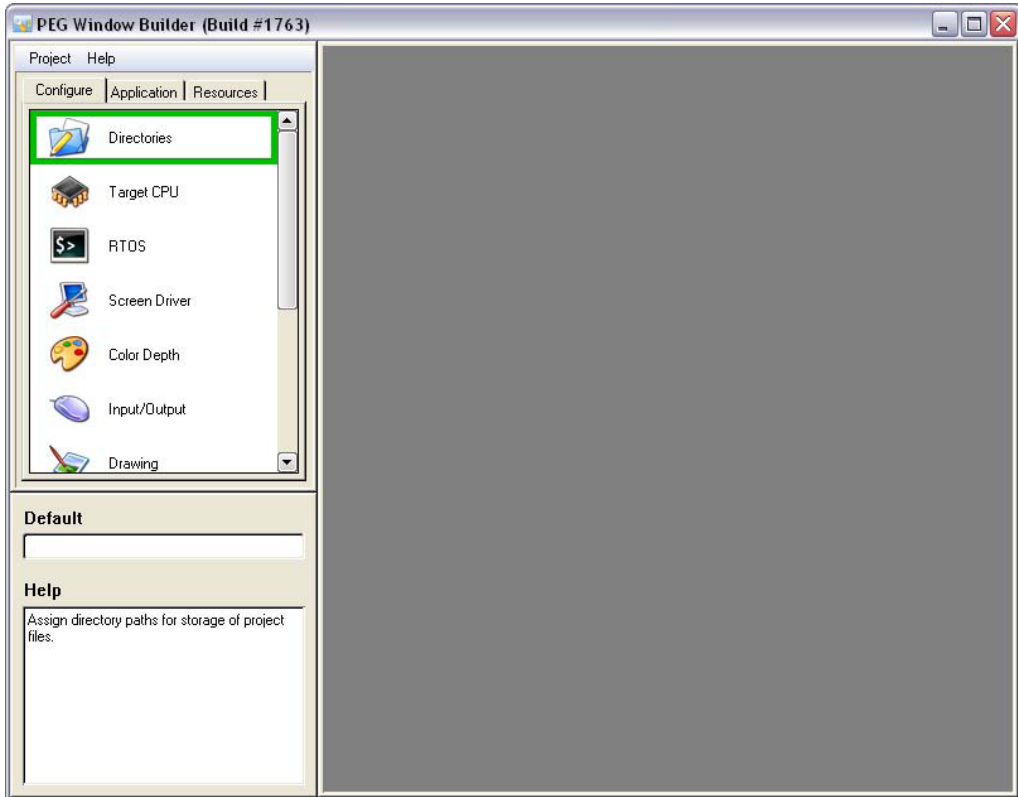
prototypes, message definitions, control IDs, string IDs, and other definitions required for your application software to compile and run.

Some of the source files produced by WindowBuilder are designed to be edited or modified outside of the WindowBuilder environment. These are the screen files or modules that define each of your display screens. Later in this manual we will describe in detail how these files can be edited by you and also updated by WindowBuilder ***without losing any of your changes.***

Other files generated by WindowBuilder, specifically the configuration and resource files, should never be edited by hand and should only be modified by regenerating them with WindowBuilder. It is very important as you read this manual to remember which files are 'hands off' and which files can be edited by hand as well as being updated or modified by WindowBuilder.

1.1.5 Screen Layout

When you run WindowBuilder, you will see a screen similar to the screen shown below. This is the default appearance of the PEG WindowBuilder application.



The WindowBuilder environment contains three main windows or screen areas. We refer to these windows as the **Project** window, the **Properties** window, and the **Target** window.

The **Project** window is the upper left display window. This window provides a high-level view of the information in your project. The Project window appearance changes form depending on which of the three mode tabs is selected, but it is always a high-level view of various parts of your application.

The **Properties** window is the lower-left display area. This window normally allows you to change the properties associated with the selected item.

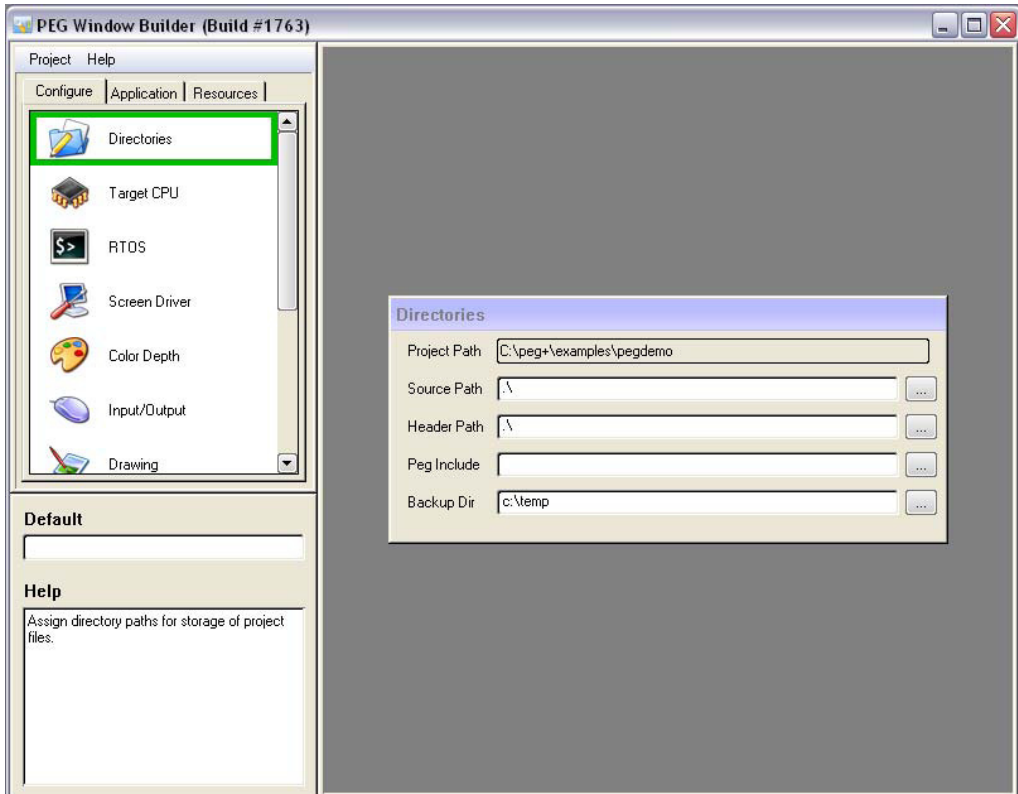
The **Target** window takes various forms depending on the operating mode. In general, the Target window displays detailed information related to what has been chosen in the Project window.

1.2 Operating Modes

There are three general modes for the WindowBuilder application, chosen by selecting one of the three tabs on the Project window. These modes are Configuration mode, Application mode, and Resource mode.

1.2.1 Configuration mode

Configuration mode is chosen by selecting the 'Configuration' tab at the top of the project view window. Configuration mode looks like this:



In configuration mode, you are configuring WindowBuilder and your PEG library according to the requirements of your target system. This is where you select the screen driver, color depth, screen x,y dimensions, and other settings for optional features that can be included in your PEG library.

When you make configuration changes it immediately affects the operation of WindowBuilder. For example, if you configure for 16bpp

(65K) colors, all the images used in your application are immediately resampled and converted to 16-bpp color depth.

Whenever you make configuration changes, you must use the Project|Generate|Configuration Header menu command to regenerate the pconfig.hpp header file that is included by all of your PEG library source files. You must also rebuild the PEG library for those configuration changes to take effect on your target system. If your application looks correct in the WindowBuilder environment, but does not look correct when running on your target system, this is often caused by failing to generate the pconfig.hpp configuration file or failing to rebuild the PEG library.

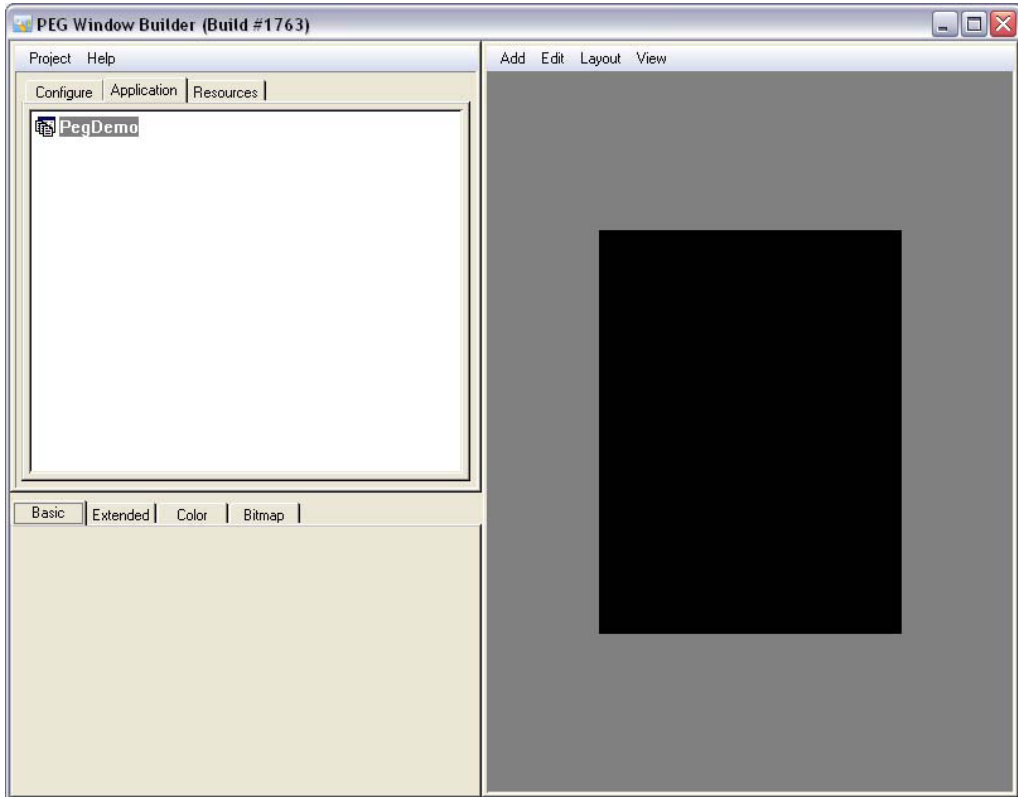
When you create a new project, you will generally want to run through each setting in configuration mode before moving on to defining your application screens or application resources. For this reason, WindowBuilder always starts in Configuration mode when you create a new project.

Each configuration setting is described in section 3.3.

1.2.2 Application Mode

Application mode is selected by clicking on the Application tab of the Project view window. Application mode looks like this:

PEG WindowBuilder™



Application mode allows you to create screens, lay out the child elements of each screen, define screen navigation, and perform a large number of additional operations all related to creating and modifying what your application will display.

The Project view in this mode displays a tree structure representing each of your screens and each of those screens child controls.

When in the application mode, the Target window displays a WYSIWYG screen of your application. The Target window is actually creating, modifying, and drawing PEG library objects as you edit the

screen. Screen editing in general is done using typical drag-and-drop editing techniques within the Target window.

The lower-left window in the Application mode becomes an 'object properties' window. This window allows you to quickly and easily change the style, font, border, ID, etc. of any graphical object. Details of working in Application mode are described in a later section of this manual.

1.2.3 Resource Mode

Most, if not all, PEG application programs utilize various images, fonts, colors, and string data as part of the application. PEG commonly refers to all of these as **Resources**. Resources are external to the application software itself, but are called on when needed. Resources are referenced by Resource ID, and are managed at run time on your target by the PEG Resource Manager. Resources can be changed, both at compile time and at run time, without any modification to the application software.

A large part of implementing any graphical application is defining and organizing the resources used by that application. WindowBuilder resource mode is implemented to accomplish exactly that. In Resource Mode you can create fonts, organize images, define color schemes, define any number of languages in which your software will run, and generate a **Resource File** that will be linked to your application to make these resources available at run time.

Resource Mode is selected by clicking the 'Resources' tab in the project view window. In this mode you can add, delete, modify, and generate resource files for your project resources.

1.3 Configuration Settings

In this section, we will detail each of the PEG library configuration settings. Because each of the PEG software libraries are

PEG WindowBuilder™

continuously improving and updating, your configuration screens may not look exactly like those displayed here. Also, certain settings are supported only in certain versions of the PEG libraries. For this section, we will describe the most complete set of configuration settings which are those for the PEG Pro library. If you are using the C/PEG or PEG+ library, some of these settings will not be available to you.

1.3.1 Directories

This dialog allows you to assign the directory path into which WindowBuilder will write generated files. Directory names can be absolute or relative to the project location. The default directories when you create a new project are '.', which means all files are written to the same directory as the project itself.

Source Path

The directory into which WindowBuilder will write source (.cpp) files.

Header Path

The directory into which WindowBuilder will write header (.hpp) files. This can be the same as the Source Path.

PEG Path

The directory into which your PEG distribution was installed. This is used to locate the pconfig.hpp header file when configuration changes are made.

Backup Dir

The directory into which WindowBuilder will back up your project file. Whenever your project is saved, WindowBuilder creates a backup of the previous project in the Backup Dir.

1.3.2 Target CPU

This section allows you to define your target CPU. Certain PEG screen drivers and input drivers are modified depending on the target CPU type. If your CPU is not listed, simply choose 'other.' Also, select if your target CPU is BIG (i.e. 68K) or LITTLE (i.e. x86) endian.

1.3.3 RTOS

The **Desktop Environment** setting allows you to define the environment you are using for desktop development. The main desktop development environments are Windows and Linux/X11. There are also desktop simulator versions of several supported RTOSs.

The **Target Environment** tells PEG which RTOS you will be running with on your target, including standalone operation (no RTOS).

The **Ticks Per Second** setting specifies how often the PegTimerManger::TimerTick function is called. As delivered, most RTOS integrations call the PegTimerManager::TimerTick function 20 times / second, for a resolution of 50 ms. This setting is exported as the ONE_SECOND definition which is used by PEG objects and may be used by your application for timing purposes. If you modify your RTOS integration and change the frequency of the call to TimerTick, you will need to modify this setting.

PEG WindowBuilder™

Priority is the PEG task priority. This can be a numeric value or an RTOS-defined priority definition. This setting should match your RTOS priority definition methodology.

The value of this setting is up to you. As a general guide, PEG should be set to run at a higher priority than noncritical tasks, but a lower priority than critical real-time tasks.

If you are running PEG standalone, the Priority setting has no effect.

Heap Size is the size of dynamic memory heap. This setting only has an effect if you are using the PEG heap manager.

Stack Size is the PEG task stack size.

1.3.4 Screen Driver

This dialog allows you to specify both your desktop and target screen driver and various options that may be supported by that screen driver.

Desktop Screen Driver specifies the color depth you will be running in your desktop environment. It should match your target system color depth for WYSIWYG operation.

H-Res and **V-Res** are the screen physical x,y dimensions in pixels.

Screen Rotation indicates whether your screen is mounted in native orientation or rotated from the native display mode. If you are using an LCD display that is natively 320W × 240H pixels, but you mount the display such that the user observes 240W × 320H pixels, you must select a screen rotation. If the screen is mounted in native orientation, select None.

Target Environment specifies the screen driver you will be running on your target hardware. It might be the same driver you are using on

the desktop, but most commonly it is a driver written specifically for your target hardware.

The drop-down list allows you to select from a list of drivers that were installed as part of your PEG package, which is determined by your installation key.

If the screen driver you intend to use is not shown in the drop-down list, or if you have created your own driver from one of the PEG driver templates, select 'custom' and type the name of the driver header file in the edit field to the right of the custom button.

The additional settings at the bottom of this dialog are used to configure options of your screen driver. Refer to your screen driver documentation to learn which options are supported in the screen driver you intend to use.

1.3.5 Color Depth

This dialog allows you to specify the color depth of your system and various color format options.

1.3.6 Input/Output

This dialog allows you to specify what type of I/O devices are supported on your target system. A keypad, touch screen, or mouse are common input devices. If your target system supports a file system, PEG can use that file system for loading resources. In that case, turn on File I/O support and the type of file system.

File Dialog and **Directory Browser** are specialized PEG classes for displaying file and directory information. If your system supports File I/O, you can turn on these options to include these classes in your PEG library.

1.3.7 Drawing Options

These settings configure support for various drawing functions and specialized classes. You can turn these on or off as desired to meet the requirements of your application.

The Horizontal and Vertical scroll bar configuration dialogs allow you to specify the default appearance of scroll bars that are automatically placed on PegWindow-derived display objects when scrolling is enabled.

The padding and whitespace settings are defaults used to position text on various PEG library control types.

1.3.8 Image Conversion

This dialog allows you to specify whether or not various *runtime* image decoders are included in your PEG library. The run time image decoders allow you to process JPG, GIF, BMP, and PNG images read from an external source and display them within your PEG application. If all of your images are included as resources, and generated by WindowBuilder, you do **NOT** need the run-time image decoders.

1.3.9 Fonts

This dialog allows you to specify the default font used by various control types. As part of the source code generation process, WindowBuilder will assign custom fonts to your controls when needed. This dialog allows you to specify the default font used for each object type, which allows WindowBuilder to recognize when a custom font assignment is required.

1.3.10 Strings

This dialog allows you to specify how strings are to be encoded in your application and whether or not PEG will provide the common string API functions (strcpy, strlen, etc.).

If you are creating a multilingual application, you will want to enable **Unicode** support unless all of your languages characters are in the Latin and Latin 1 code Unicode code pages.

When Unicode support is enabled, your strings can be encoded as raw 16-bit Unicode or they can use UTF8 encoding. UTF8 encoding can marginally reduce the const data size of your string table if you have a large amount of string data. If the size of your string table is not a concern, you will have slightly better runtime performance if you use raw Unicode encoding.

The LTOA function is a non-ANSII run-time library function utilized by PEG. PEG optionally provides its own implementation of this function. If your compiler provides the LTOA function, you can disable the PEG implementation by turning off this option.

The remaining options include or exclude support for localized (non-Unicode) string encodings.

Miscellaneous

This dialog is a catch-all for configuration settings that do not fit well in other categories.

The **User Defined** options allow you to type any number of additional #defines that will be included in your pconfig.hpp header file. You can use this to create your own conditionals that will be utilized in your application software. The data that you type in this edit box will be appended to the end of pconfig.hpp header file, so normal C++ syntax rules apply.

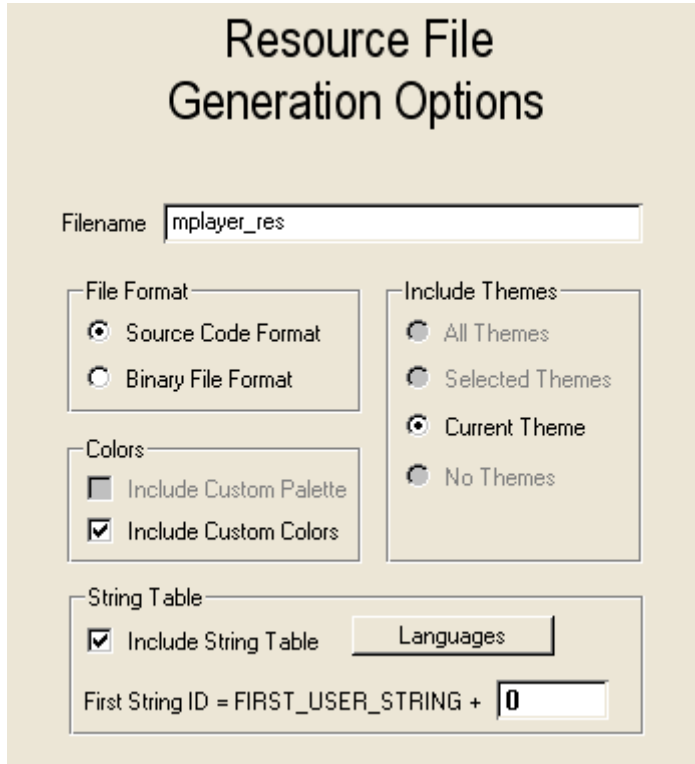
1.4 Working with Resources

There are four types of resources: strings, bitmaps, fonts, and colors. Resources are referenced in your application software using Resource IDs. This makes it possible to modify your resources and change the appearance of your user interface without modifying your application software.

Resource IDs are prefixed with a three character identifier to make it easy to identify each resource type. The ID prefixes are **SID_**, **BID_**, **FID_**, and **CID_** corresponding to **String ID**, **Bitmap ID**, **Font ID**, and **Color ID**.

WindowBuilder is used to create and manage your resources. Even if you choose not to use WindowBuilder for screen design and layout, you will still use WindowBuilder to create and manage your application resources.

To view and modify your project resources, click on the Resources tab of the project view. When you click on the top node of the resource view, you will see the Resource Generation dialog:



The dialog box is titled "Resource File Generation Options". It features a "Filename" field containing "mplayer_res". Below this are four main sections: "File Format" with radio buttons for "Source Code Format" (selected) and "Binary File Format"; "Colors" with checkboxes for "Include Custom Palette" (unchecked) and "Include Custom Colors" (checked); "Include Themes" with radio buttons for "All Themes", "Selected Themes", "Current Theme" (selected), and "No Themes"; and "String Table" with a checked "Include String Table" checkbox, a "Languages" button, and a text field for "First String ID = FIRST_USER_STRING + 0".

This dialog is used to specify what is included in your resource file and various other options.

The File Format specifies the format of the resource file that will be generated. Most commonly you will use the Source Code Format resource file. This produces a C++ file that you will compile and link with your application files. You can also produce a Binary resource file.

A Binary resource file is used when you want to support the installation of new resources at run time. For example, you may ship your product supporting only the English language. You can then add

PEG WindowBuilder™

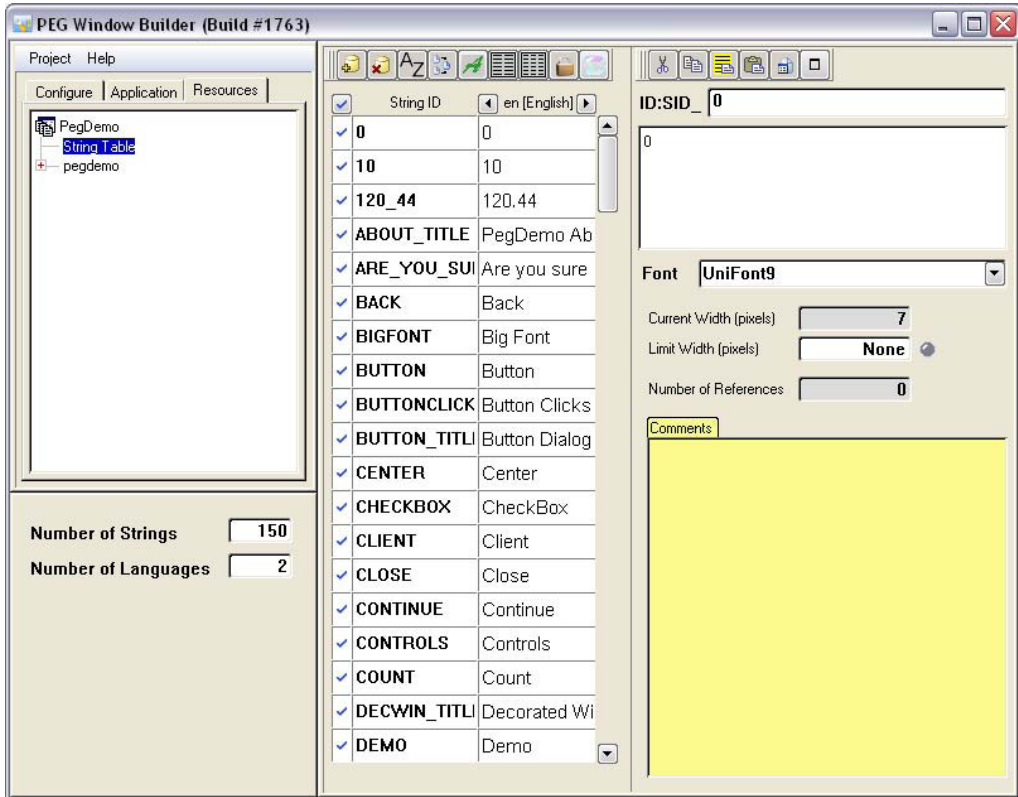
additional languages by generating a Binary resource file containing Spanish, Russian, and French. You can then install this binary resource file on your running system (assuming your target has a file system).

As another example, you might produce Binary resource files containing new 'Themes' (themes are described in a later section) and allow your customers to download those themes from your company website.

1.4.1 String Resources

PEG applications use a table of strings, fittingly called the StringTable, to hold all constant string literals. Even if your application will initially only support one language, it will still use a string table. This makes it easy to extend the application to multiple languages at any time.

The string table is managed by the String Table Editor. You open the String Table Editor by clicking on the Resources tab of the Project View, then click on the String Table node of the project. You will see a screen similar to this:



You can maximize the StringTableEditor within the window by clicking on this button:



The string table editor displays string ID, string literal, and various other information related to each string. The toolbars at the top are used to add new strings, remove strings, and configure the number of languages included in the string table.

PEG WindowBuilder™

The table can be viewed in two column mode (StringID and one Language) or three column mode (StringID, Reference Language, and Second Language).

The Notes field allows you to enter descriptive text associated with each string. This text is included in the string table export to assist translators in making accurate string translations.

The Current Width and Limit Width fields indicate the string width in the chosen font, and an optional limit to aid translators in making translations that will fit within the control to which the string is assigned.

The Number Of References field indicates how many times that string is used in the application, at least that part of the application that has been created within WindowBuilder.

Once you have added custom fonts to your project, you can choose any of those fonts to display the table itself and each individual string. This will be necessary once you add strings that cannot be displayed using the Latin character pages, which are the only characters included in the two PEG default fonts.

Whenever you make a change to the string table, it is important that you regenerate the resource file. The resource file contains all of your string table data (along with other resources). To generate your resource file, click on the top node of the Resource view, and click on the Generate Resource File button.

1.4.2 Themes

Themes are collections of bitmaps, colors, and fonts. Themes are used to change the look of your application at run time.

Working with Resources

Bitmap, font, and color resources are organized into themes. Your application will usually begin with one theme. You can add additional themes at any time.

When you create a new project, it will have one theme named 'Default Theme.' You can give the theme any name you prefer by clicking on the theme name and typing a new name in the theme properties window in the lower left.

To create a new theme, you select the current theme by clicking on it in the project view and clicking on the 'Clone Theme' button in the lower left, or by right-clicking on the theme and using the Clone Theme menu command.

When you clone a theme, you are creating an exact copy of the current theme. You can then modify the bitmaps, colors, and fonts in the cloned theme to create a new theme. If your project has multiple themes, any additions or deletions you make in one theme are automatically reflected in the other themes. In other words, all themes are forced to have exactly the same number of bitmaps, fonts, and custom colors.

Within each theme, there are nodes for image groups, colors, and fonts. You can create as many image groups as you like within each theme. Image groups are used to organize your bitmaps into different folders, because a project may contain several hundred images and it becomes important to organize those images into different groups to make it easy to locate them. For example, you might create an image group named 'Backgrounds' to contain various wallpapers or window backgrounds, another image group named 'buttons' that contains various button images, and a third bitmap group named 'icons' to contains static display icons.

When you click on a theme, the lower left properties window display information about that theme. The 'Include Theme in Resource File' check box is an important selection. You can opt to include or

exclude any theme from your resource file. This feature is used to create run-time installable themes. We will describe this more in a later section.

1.4.3 Bitmap Resources

Bitmaps or images are another type of application resource. Bitmaps are added to image groups. Once you have added a bitmap to your project, you can assign that bitmap to any PEG display object that supports bitmap drawing.

To add new bitmaps to your project, click on the desired Image Group and use the right-click menu or the button labelled 'Add New Image To Group.' This brings up a file browse dialog that allows you to select any JPG, GIF, PNG, or BMP type source image.

Once you select a source image, WindowBuilder reads the source image and color-converts the image to your target color depth. The image can be displayed by clicking on the image ID name.

When you select a specific image within an image group, the lower-left properties window displays the image properties. You can change the image name, bitmap id, and conversion settings.

Whenever you modify your bitmap resources, you must generate a new resource file to contain your new bitmap data.

1.4.4 Font Resources

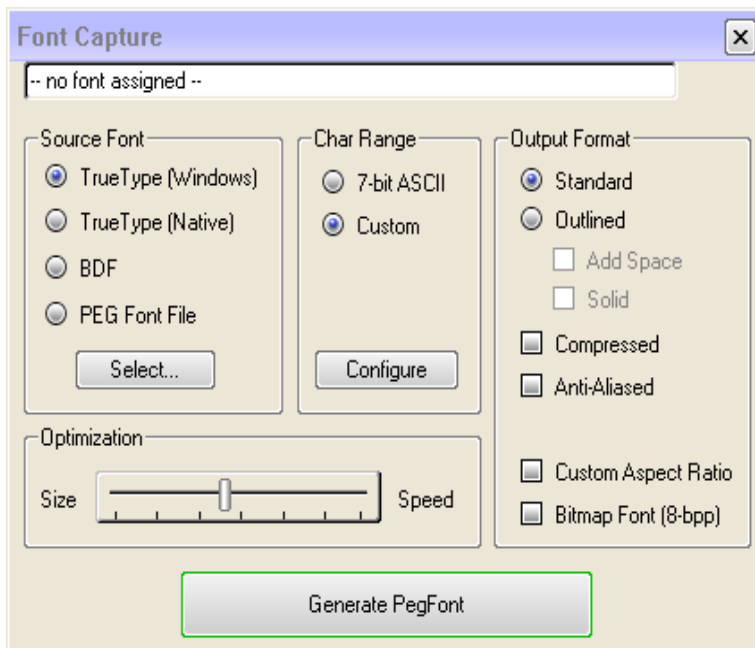
Fonts are also application resources. You can create new fonts and add new fonts to your project by clicking on the Fonts node of any Theme in the Resource View.

Creating fonts and adding fonts to your project are two distinct and separate operations. Fonts are created by running the FontCapture utility within WindowBuilder and saving the fonts you create to your hard drive as 'Peg Binary Font' or '.pbf' font files.

Fonts are added to your project by clicking the Add PegFont to Theme button or right-click menu command. You can only add *.pbf PEG binary fonts to your project.

Step 1: Creating a new font

To create a new font, click on the Fonts node within any Theme and click on the Create New PegFont button. You should see a screen similar to this:



PEG WindowBuilder™

This is the FontCapture utility. FontCapture allows you to create any number of PegFonts from any TrueType or BDF font file. You can also open and edit previously-created PegFonts within this utility.

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 select TrueType 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 BDF source font format, the Select button allows you to select the actual .bdf file from your development station file system.

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 multi-lingual 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 between normal bitmapped font output, outlined font format, or anti-aliased font format. 'Normal' format is a 1-bpp on-or-off format. This format requires the least storage space, but also produces the least readable font. The Outline format is a 2-bpp format, where each character is outlined in a color different than the character fill color. The anti-aliased font format is a 4-bpp grayscale format. This format requires the most disk space, but produces the most readable text on

your system. Anti-aliased fonts can be drawn in any foreground/background color combination.

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 that the pixel should be the foreground color, bitmap value 1 indicates that the pixel should be in the outline color, and bitmap value 2 indicates that 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.

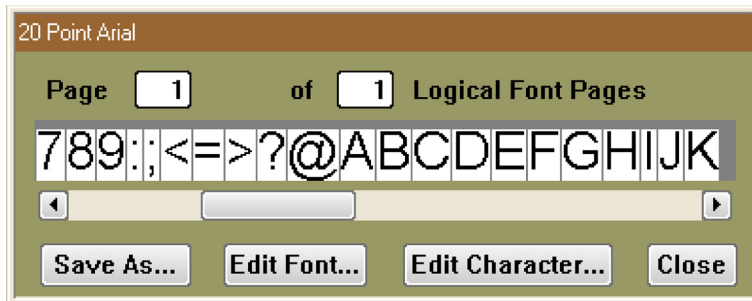
The **Binary** output format is used when you want to create a binary PegFont file, rather than the more common C source file.

The **Solid** and **Add Space** check boxes 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 **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

PEG WindowBuilder™

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, 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. 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 file of your choosing.

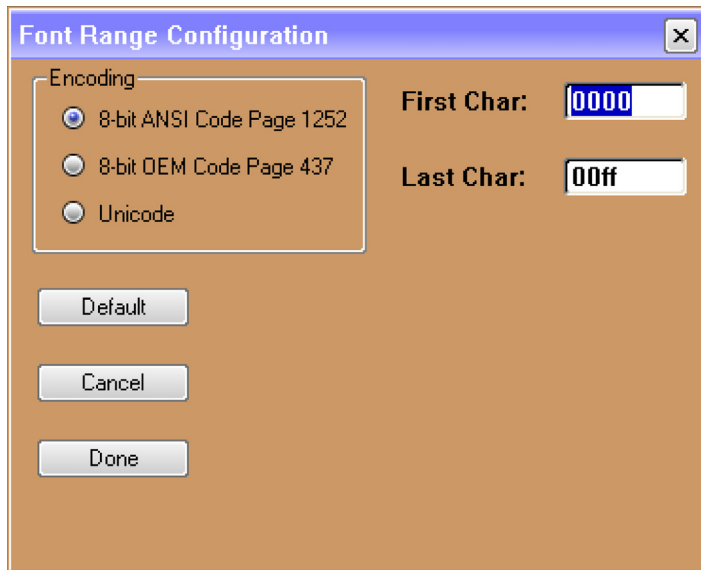
1.4.5 Configuring Character Range

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 that will be recorded in the output file.

It is often the case that a particular font is only used to display a certain range of characters; for example you may define one font that

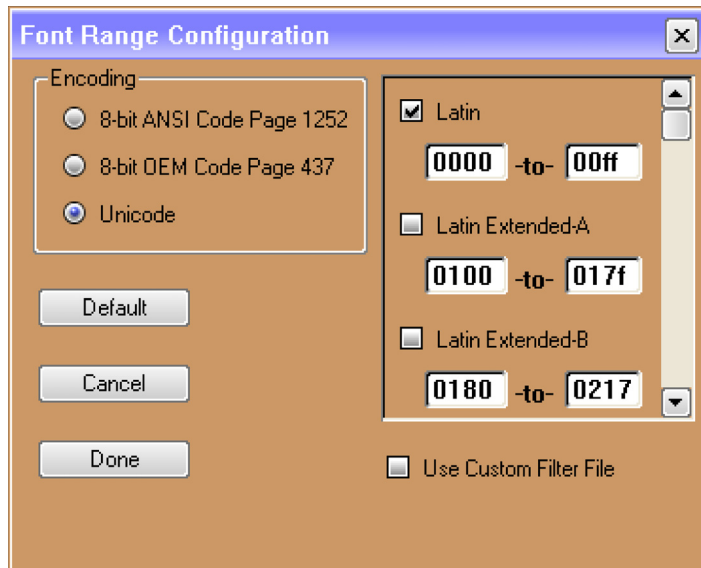
will be used only for displaying numbers. In this case, you do not need or want to encode the entire ASCII character range in the output file. Instead, you can enter a limited character range by selecting the 'Custom' button and entering the range of characters in the Range Configuration dialog, shown 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 a significant amount of memory over capturing the entire ASCII character set.

1.4.6 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.

What is UNICODE?

If you are a software developer from North America, you may only be vaguely familiar with UNICODE and what it means for your software. UNICODE is a standard definition of 16-bit character encoding that encompasses 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).

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, 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.

PEG WindowBuilder™

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 attempt 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 want 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.

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. 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 table 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.

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.

If the number of characters contained in ANY single font file exceeds 256, you will need to run PEG in UNICODE mode, meaning that all PEG strings will be encoded using 16-bits/character.

The advantages of the UNICODE approach are that 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 into UNICODE.

1.4.7 Color Resources

Colors are the final type of application resource. To view and edit colors, you select the Color node within any theme in the Resources view.

Within the Color node, you will find three color groups. These are the fixed colors, default colors, and custom colors.

PEG defines a fixed set of 16 'VGA Palette colors.' These colors cannot be changed or modified. These are the Fixed colors.

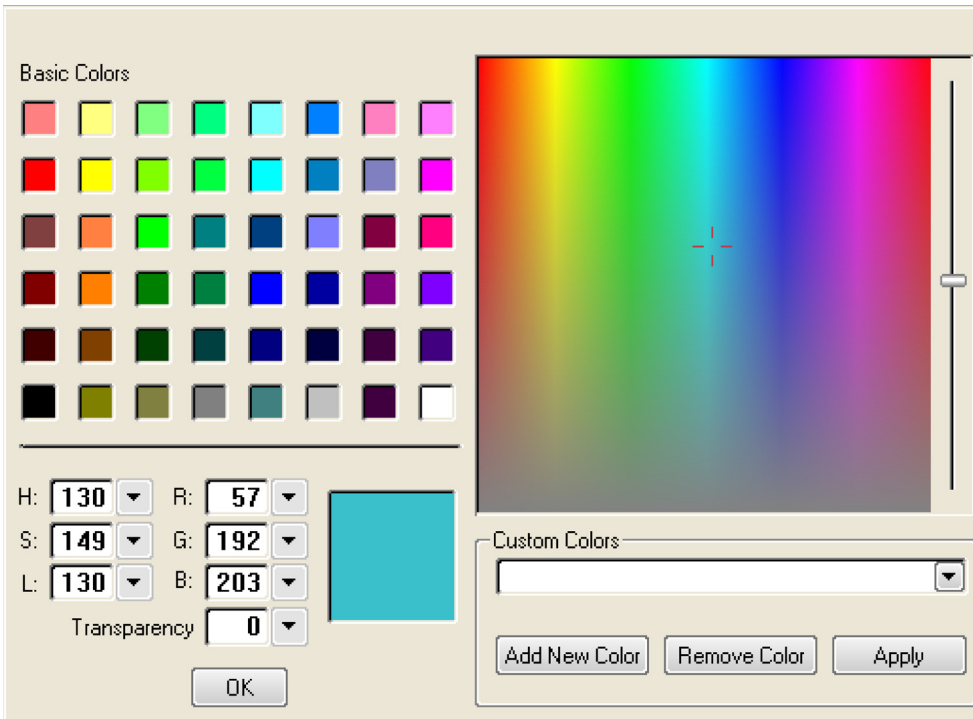
The Default colors are colors PEG widgets use as defaults if you do not specifically assign a color to a particular object. There are a large number of default colors corresponding to things like window fill color, button fill color, scroll bar fill color, scroll bar thumb color, etc. You can change any of these default colors to your requirements.

Changing the default colors is more efficient than assigning a custom color to every object, both in terms of code size and, marginally, in terms of run-time performance. PEG objects will always initialize themselves to use the default colors. If you specify a custom color for particular object, this default is then overridden to use your custom color. Of course, you will nearly always need to assign particular colors to particular objects at one point or another, which is easily done in the WindowBuilder environment.

To edit any default color, simply right-click on the color thumbnail to bring up the color editor (described below).

Custom Colors

Custom Colors are colors you define for your application to use. To define a custom color, click on the Custom color node within a color group, and click on the Edit Custom Colors button in the properties window. You will see a screen similar to this:



This is the color editor. If you invoked the color editor by clicking on an existing color, it will initialize to the selected color value.

To add a new color, click the Add New Color button and type a color ID name. Colors use IDs just like all other application resources. After typing the color ID name, hit 'enter.' You can now use the controls of this editor to define your color.

PEG WindowBuilder™

You should use Color ID names that describe where the color is used, rather than describing the 'look' of the color. For example, ColorID names like 'BUTTON_FILL' and 'MENU_TEXT' are good color names. ColorID names like 'VERY_LIGHT_GREEN' are not good ColorID names. The reason for this is that if/when you create multiple themes for your application, you will very likely change some or all of your custom colors for each theme. However, the ColorID will not change from one theme to another. Therefore, the ColorID should indicate where and how the color is used, not the appearance of the color.

Clicking on one of the Basic Colors thumbs initializes your new color to some known starting point. You can then adjust the hue, saturation, luminance, or red/green/blue values of your color to create the exact color you require.

The transparency value is only used if you are running with 16-bit + Alpha or 24-bit + Alpha screen driver. In this case, Transparency value 0 is fully transparent, and transparency value 255 is opaque. You can set the transparency value for your color to any value within this range.

1.4.8 Generating Resource Files

Whenever you make changes to the fonts, colors, string, or bitmaps included in your project, you need to generate a new resource file. This new resource file will either be compiled/linked with your application (a Source format resource file) or installed on a running application (a Binary format resource file).

To generate a global resource file containing all Themes, click on the top node of the resource view in the Resources view window to display the Resource File Generation Options. To generate a resource file that contains only one theme, click on that theme name in the Resource View.

The resource file is by default given the name of your project, with ‘_res’ appended. When you generate a source format resource file, both a .cpp and a .hpp file will be created. Note that any previous resource file of the same name is overwritten. For this reason, you should **never edit your resource file by manual editing.**

Click on the items you want to include in your resource file, and click on the Generate Resource File button.

Each Theme and Image Group have their own include/exclude checkbox. A Theme or ImageGroup will NOT be included in your resource file unless the “Include In Resource File” checkbox is checked.

If you generated a source-format resource file, you simply need to build your application using the new resource file. If you generated a binary resource file, the PEG API function ‘LoadBinaryResourceFile’ is provided to install the binary resource file into your running application.

1.5 Application Mode

The third WindowBuilder operating mode is Application Mode. Once you have configured your target and (optionally) added resources to your project, you are ready to begin creating your graphical application. Application mode allows you to easily and quickly design your screens and child controls that will become your application.

You enter application mode by clicking on the Application tab of the Project View.

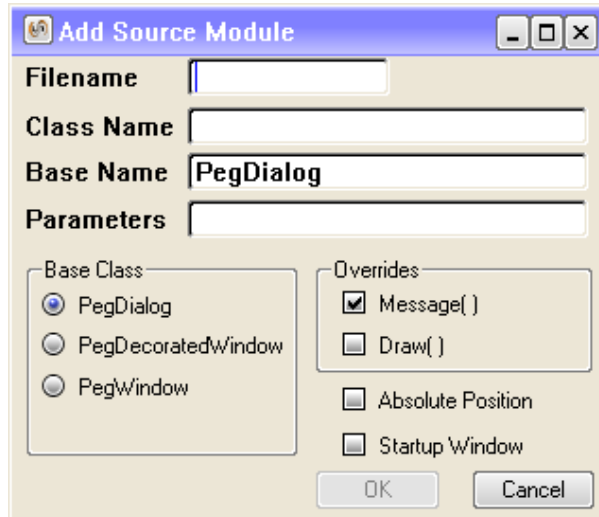
1.5.1 Creating Modules

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.

In Application Mode, the Project View window displays a PegTreeView depicting each of the modules included in the current project. Each module is listed in alphabetical order by class name. Each top-level node of the PegTreeView control represents a top-level class constructed with WindowBuilder. If you expand a top-level node, you will see the corresponding source and header file associated with that top-level class. If you select a source module by left clicking with the mouse, the Target window will display the PEG objects defined within that source module.

The Target window always operates on the selected module. If no module is selected, none of the Target window editing commands are operational.

At any time, you can 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 **ClassName** field allows you to specify a name for the PegWindow-derived class you are creating, i.e. the name of the class which will define the window or screen you are developing.

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 want 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.

PEG WindowBuilder™

The Startup Window check box specifies that this window will be the first displayed when your application executes. When this check box 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.

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 which PEG library class will be the base class for your new Window. This will usually be PegDialog, but could also be PegWindow or PegDialog. If you have created your own derived class to serve as the base class for this window, you can enter the name of that class in this field.

The **Overrides** group is used to tell WindowBuilder which function of the base class will be overridden by the class you are defining. Only two options are supported by WindowBuilder (although you can of course add your own function overrides to the completed class: the Message function and the Draw function. 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 postponed; i.e., there are no left-top incoming parameters and the window and child controls use absolute pixel positioning.

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 source file 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 that 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 on the module name in the Project tree view. This will bring up a dialog window allowing you to change the module name, file name, and other module parameters.

Note that you are not allowed to delete sub-nodes from the PegTreeView displayed within the source page of the notebook. This is accomplished by individually deleting objects from within the Target window, described below.

In order to begin designing your application screens, you must create and add at least one module to your project.

1.6 The Target Window

In Application Mode, the 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.

PEG WindowBuilder™

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 view window. If no modules are included in your project, you must first create a new 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.

1.6.1 Selecting Objects in the Target Window

Objects in the Target window are selected and edited using the mouse. When you click on an object in the Target window, a red border is drawn around the object to indicate that it 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.

When you select an object, the object's properties are displayed in the lower-left Properties window. You use the properties window to assign fonts, give your objects IDs, assign bitmaps, and perform various other properties setting for each object type.

1.6.2 Target Window Menu Commands

The target window menu commands always operate on the currently 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:

- TextButton
- MLTextButton
- BitmapButton
- DecoratedButton
- CheckBox
- RadioButton
- SpinButton
- Icon

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|Text

PEG WindowBuilder™

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

- Prompt
- VertPrompt
- String
- TextBox
- EditBox

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|Indicator

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

- Finite Dial
- Bitmap Dial
- Light
- Bitmap Light
- Linear Scale
- Bitmap Scale

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|Slide/Scroll

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

- Slider
- VScroll
- HScroll

Note that 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 that are used to contain or group other child gadgets. These include:

- PegGroup
- ComboBox
- VertList
- HorzList

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

Edit|Properties

This command invokes a properties dialog for the selected object. One and only one object must be selected in order for this menu command to be active. ***You can also invoke the edit properties dialog window by right-clicking with the mouse on the selected object.***

The properties dialog is context sensitive depending on the type of object that 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 that 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' that 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 objects 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

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

Layout|Equal Width

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 hot spots (described below), you can click on the product hot spots 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 ensure that 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

PEG WindowBuilder™

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 that will produce input messages into the PegMessageQueue. A common example would be a product that provides the end user with up/down/right/left and select type navigation 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 WindowBuilder. In this mode, you can click on the product background image hotspots to navigate through your UI. This allows you to fully exercise 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.

1.6.3 Generating Application Source Code

WindowBuilder produces 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 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 WindowBuilder.

You generate your application source code by either right-clicking on the module name in the Project view and selecting the 'Update Current Source Module' command, or by using the Project|Generate|Application command to update all modified source modules.

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 described below.

When you instruct PEG WindowBuilder to produce/update the source files, WindowBuilder first looks to see if the source file already exists. If it does, WindowBuilder enters '**Merge Mode.**' In Merge Mode, WindowBuilder is very careful not to lose any of your custom modifications.

A snippet of the source module produced by WindowBuilder may look like this:

```
/*-----*/
/*-----*/
// Source file originally created by PegWindowBuilder
//
// Class Name: MainScreen
//
// Copyright (c) Swell Software, Inc.
// www.swellsoftware.com
//
// All Rights Reserved
//
// Notes:
/*-----*/
/*-----*/
```

PEG WindowBuilder™

```
/* WB Auto-Generated Start (1) */
#include "peg.hpp"
extern PegResourceTable Default_Theme_ResourceTable;
/* WB Auto-Generated End (2) */

/*-----*/
/*-----*/
/* WB Auto-Generated Start (3) */
void PegAppInitialize(PegPresentationManager *pPresent)
{
    PegResourceManager::InstallResourcesFromTable(
        &Default_Theme_ResourceTable);
    MainScreen *pWin = new MainScreen(0, 0);
    pPresent->Center(pWin);
    pPresent->Add(pWin);
    /* WB Auto-Generated End (4) */
}

```

To avoid losing your changes, never make any manual edits between the markers WB Auto-Generated Start and WB Auto-Generated End.

These Auto-Generated Start and Auto-Generated End markers are placed in strategic locations within the source file produced by WindowBuilder. You can make any changes you like to the source file, as long as you do not make changes within a section delimited with these start and end markers.

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 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 that you have done will be preserved entirely during the source file update process.

1.6.4 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 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 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 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 WindowBuilder that uses an implicit pointer:

```
Add(new PegPrompt(ChildRect, "Text"));
```

Note that 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 WindowBuilder.

Temporary Pointers

Next up in the WindowBuilder source code generation process is the temporary pointer. This type of pointer is used by WindowBuilder

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. 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
```

WindowBuilder will always use the generic names pChild x for temporary automatic pointers. 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 WindowBuilder will create and use 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 **unchecked** the 'Member Pointer' box. Note that 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 WindowBuilder in ways that are not supported by the 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 Auto-Generated End */”* 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

PEG WindowBuilder™

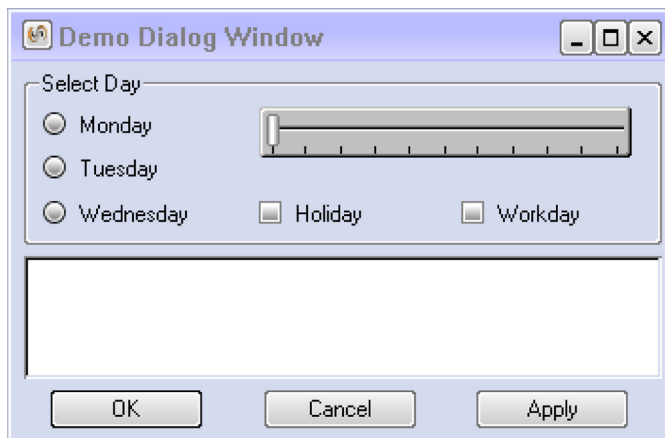
constructor and used at all times to reference the child object. You can instruct WindowBuilder to create a member pointer for a child object by checking on the 'Member Pointer' checkbox in the properties dialog and typing a name in the 'Pointer Name' field. Note that the name must be a valid C++ variable name or your compiler will flag an error when you compile the generated module (no name checking is done by WindowBuilder!).

1.7 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':



Example 1: Creating a simple PegDialog window

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

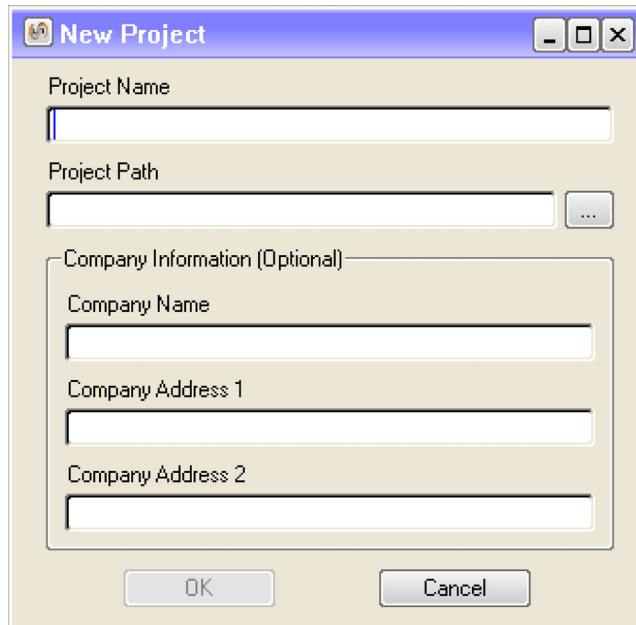
1.7.1 Creating and Configuring a Project:

Under MS Windows 95/98/NT/2000, start the PEG WindowBuilder executable program, **pwinbld.exe**. :

For this example, we want to begin a new project, so you should select the **New Project** button on the startup screen.

➤ Step 1- Configure Project and Directories

Whenever you begin a new project, 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, 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.'**

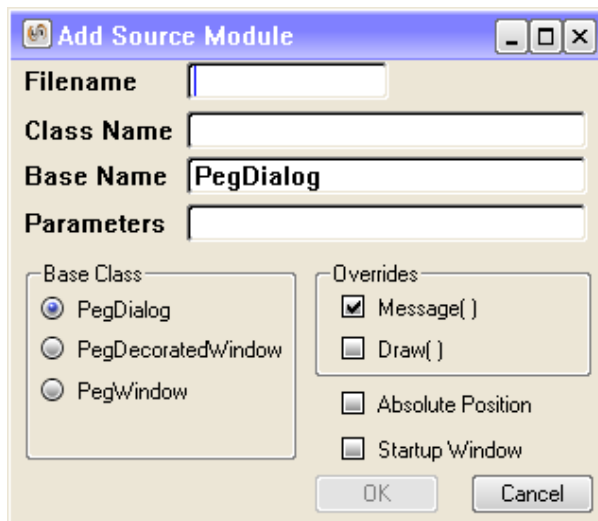
You will next want to run through the configuration dialogs, specifically the Directories dialog, to configure where your source and header files will be written to your computer hard drive.

Example 1: Creating a simple PegDialog window

When a new project is created, the target window begins to display the target screen. The default target screen resolution is 240 × 320 pixels with 16-bit (65K) colors. This can be modified to any supported color depth and resolution using the Configuration dialogs.

➤ Step 2- Add a new Module

You are now ready to create a new module. Each WindowBuilder module contains a unique class declaration and class implementation. Click on the Application tab in the Project view, and 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

PEG WindowBuilder™

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 that will display our example window. Leave the remaining dialog fields at their default values and select 'OK.'

You will now see that your project tree contains a new node, with the name DemoDialog. When child objects are added to the dialog they will be listed underneath the module name in the tree view.

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

1.7.2 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 that 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. Note that if the dialog is the same size as the target screen, you will not be able to move it anywhere by dragging it. You can only drag it within the boundaries of the target screen.

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

Example 1: Creating a simple PegDialog window

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

Click on the dialog window in either the Project window or the Target window to insure it is selected.

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.

➤ Step 5- Add a PegGroup to the Dialog

Make sure that 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 that it is similar in position and size to the reference example. Edit the group properties by going to the Extended properties page and enter

'**Select Day**' in the Text field. This assigns the text value that is displayed as the group title.

➤ **Step 6- Add Radio Buttons to the Group**

Once you have the PegGroup in position, ensure that 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 by right-clicking in the radio button client area. On the Basic properties page, enter '**IDRB_MONDAY**' in the ID field. On the extended properties page, enter '**Monday**' in the Text field. 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 arrow 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.

Example 1: Creating a simple PegDialog window

➤ Step 7- Using Layout commands.

To ensure 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. Next, hold down the <ctrl> key and left click on the 'Tuesday' and 'Wednesday' radio buttons in turn. You will see that 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 that the radio button are all exactly aligned at the left border.

Note that 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 check boxes 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|Slide/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 that 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 that 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 that the buttons are vertically aligned, and move them as a group until they are centered on the dialog window.

Example 1: Creating a simple PegDialog window

You are now done creating the dialog window!!

1.7.3 Saving Your Work:

➤Step 11- Save the project

At this point you should select the Project|Save command to save your project. This will save your work to the 'DemoDlg.wbx' project file. 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 right-click on the module name and select the 'Update Current Module' command. After the source code has been generated, PEG WindowBuilder should inform you that the source or header file has been updated.

