The PEBL Manual
Programming and Usage Guide for
The Psychology Experiment Building Language
PEBL Version 0.14

Shane T. Mueller

©2003-2014 Shane T. Mueller smueller@obereed.net

Current for PEBL Version 0.14 – May 2014
http://pebl.sourceforge.net
4 Overview of Object Subsystems 17
  4.1 Lists ....................................................... 17
    4.1.1 Growing Lists .................................... 18
    4.1.2 Recursion on lists ................................ 20
  4.2 Fonts ....................................................... 21
  4.3 Colors ...................................................... 23
  4.4 Windows .................................................... 23
  4.5 Graphical Widgets ......................................... 23
  4.6 Images ....................................................... 24
  4.7 Canvases .................................................... 24
  4.8 Shapes ....................................................... 25
    4.8.1 Circle .................................................. 25
    4.8.2 Ellipse ............................................... 26
    4.8.3 Square ................................................ 26
    4.8.4 Rectangle .............................................. 26
    4.8.5 Line ..................................................... 27
    4.8.6 Polygon ................................................ 27
    4.8.7 Bezier ................................................ 27
  4.9 Text Labels ................................................ 28
  4.10 Text Boxes ................................................. 28
  4.11 User-Editable Text Boxes ................................ 28
  4.12 Audio ....................................................... 30
  4.13 Movie Files ............................................... 30
  4.14 Custom objects ............................................ 31
  4.15 Keyboard Entry ............................................ 32
  4.16 Joystick Input ............................................ 32
  4.17 Files ....................................................... 34
  4.18 Network Connections ....................................... 35
    4.18.1 TCP/IP Overview ..................................... 35
    4.18.2 Addresses and Ports ................................ 35
    4.18.3 Sending and Receiving Data ....................... 36
    4.18.4 Closing networks .................................... 36
  4.19 Parallel Port .............................................. 36
  4.20 Serial Port ................................................. 37
  4.21 The Event Loop ............................................ 37
  4.22 Parameter Setting ........................................ 38
  4.23 Errors and Warnings ...................................... 39
  4.24 Paths and Path Searching ................................ 39
  4.25 Provided Media Files ..................................... 40
  4.26 Special Variables ......................................... 41

5 Function Quick Reference 43
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.18</td>
<td>R</td>
<td>206</td>
</tr>
<tr>
<td>8.19</td>
<td>S</td>
<td>217</td>
</tr>
<tr>
<td>8.20</td>
<td>T</td>
<td>237</td>
</tr>
<tr>
<td>8.21</td>
<td>U</td>
<td>241</td>
</tr>
<tr>
<td>8.22</td>
<td>W</td>
<td>243</td>
</tr>
<tr>
<td>8.23</td>
<td>Z</td>
<td>251</td>
</tr>
<tr>
<td>9</td>
<td>Color Name Reference</td>
<td>253</td>
</tr>
</tbody>
</table>
PEBL (Psychology Experiment Building Language) is a cross-platform, open-source programming language and execution environment for constructing programs to conduct a wide range of archetypal psychology experiments. It is entirely free of charge, and may be modified to suit your needs as long as you follow the terms of the GPL, under which the source code is licensed. PEBL is written primarily in C++, but requires a few other tools (flex, yacc) and libraries (SDL, SDL_image, SDL_gfx, and SDL_ttf) to use. In addition, a set of audio recording functions are available using the (now old and basically unmaintained) sdl_audiomix library. Finally, the waaave library optionally supports movie playback on Linux and Windows.

It currently compiles and runs on Linux (using g++), Mac OSX (using xcode), and Microsoft Windows (using code:blocks and mingw) platforms using free tools. It has been developed primarily by Shane T. Mueller, Ph.D. (smueller@obereed.net). This document was prepared with editorial and formatting help from Gulab Parab and Samuele Carcagno. In addition, much of the material in the chapter on the PEBL Test battery was contributed by Bryan Rowley. Contributions are welcome and encouraged.
Chapter 2

Usage

Most users will be able to download a precompiled version of PEBL and run experiments directly. Some advanced users may wish to compile their own version, however. The next section describes how to do this.

2.1 How to Compile

Currently, there is no automated compile procedure. PEBL requires the SDL, SDL-image, SDL-gfx SDL_net, SDL_maudioin and SDL-ttf libraries and development headers. It also uses flex and bison, but you can compile without these tools. PEBL compiles on both Linux and Windows using the free gcc compiler; on windows this is most easily supported through the code:blocks IDE. Note that SDL-image may require jpeg, png, and a zlib compression library, while SDL-ttf uses truetype 2.0.

2.1.1 Linux

PEBL should compile by typing ‘make’ in its base directory once all requisite tools are installed and the source distribution is uncompressed. Currently, PEBL does not use autotools, so its make system is rather brittle. Assistance is welcome.

On Linux, compiling will fail if you don’t have an /obj directory and all the appropriate subdirectories (that mirror the main tree.) These will not exist if you check out from CVS.

2.1.2 Microsoft Windows

On Microsoft Windows, PEBL is designed to be compiled using the Free IDE code:blocks. A code:blocks project file is included in the source code directory. Email the PEBL list for more details.
2.1.3 Mac OSX

Originally, PEBL compiled to a command-line function. Since 0.12, PEBL will compile to a .app package using xcode. An xcode package is available in the source package.

2.2 Installation

2.2.1 Linux

On Linux, there are .deb packages available for debian. However, it is fairly easy to compile and install from source. To begin, be sure that all the sdl packages are installed. Then, go to the main pebl directory and type:

```
> make
> sudo make install
```

Once installed, you can install the test battery into Documents/pebl-exp.X using the command pebl -install.

2.2.2 Microsoft Windows

In Microsoft Windows, we provide an installer package that contains all necessary executable binary files and .dlls. This installer places PEBL in c:\Program Files\PEBL, and creates a directory pebl-exp.X in My Documents with a shortcut that allows PEBL to be launched and programs that reside there to be run.

2.2.3 Macintosh OSX

For OSX, we provide a .app package that can be dragged into your Applications folder. The first time any user runs pebl, it gives the option to install the battery and other files into Documents/pebl-exp.X. Afterward, it will run the launcher from that directory.

2.3 How to Run a PEBL Program

The simplest way to run any PEBL script is via the launcher, which is available on all platforms. The launcher is covered in detail in Chapter 6. But, you can also launch experiments individually on each platform.

2.3.1 Linux

If you have installed PEBL into /usr/local/bin, you will be able to invoke PEBL by typing `pebl` at a command line. PEBL requires you to specify one or more source files that it will compile and run, e.g., the command:
> pebl stroop.pbl library.pbl

will load the experiment described in stroop.pbl, and will load the supplementary library functions in library.pbl. Additionally, PEBL can take the \(-v\) or \(-V\) command-line parameter, which allows you to pass values into the script. This is useful for entering subject numbers and condition types using an outside program like a bash script (possibly one that invokes dialog or zenity). A sample zenity script that asks for subject number and then runs a sample experiment which uses that input resides in the bin directory. The script can be edited to use fullscreen mode or change the display dimensions, for example. See Section 2.3.3: Command-Line Arguments. You can also specify directories without a filename on the command-line (as long as they end with '\/'). Doing so will add that directory to the search path when files are opened.

2.3.2 Microsoft Windows

PEBL can be launched from the command line in Windows by going to the pebl\bin directory and typing `pebl.exe`. PEBL requires you to specify one or more source files that it will compile and run. For example, the command

> pebl.exe stroop.pbl library.pbl

loads the experiment described in stroop.pbl, and loads supplementary library functions in library.pbl. Additionally, PEBL can take the \(-v\) or \(-V\) command-line parameter, which allows you to pass values into the script. This is useful for entering condition types using an outside program like a batch file. The \(-s\) and \(-S\) allow one to specify a subject code, which gets bound to the gSubNum variable. If no value is specified, gSubNum is initialized to 0. You can also specify directories without a file (as long as they end with '\/'). Doing so will add that directory to the search path when files are opened. See Section 2.3.3: Command-Line Arguments.

Launching programs from the command-line on Windows is cumbersome. One easy way to launch PEBL on Windows is to create a shortcut to the executable file and then edit the properties so that the shortcut launches PEBL with the proper script and command-line parameters. Another way is to write and launch a batch file, which is especially useful if you wish to enter configuration data before loading the script.

2.3.3 Macintosh OSX

The latest version of PEBL packaged for OSX is 0.12. It is compiled as an application bundle with both 32-bit and 64-bit architectures available. We do not support PPC architecture.

The simplest way to run PEBL is through the launcher, but you can also use Applescript to create your own sequences of experiments.
Chapter 2. Usage

On OSX, PEBL can be run as a command-line tool, just as in Linux. Once installed, the application is located at /Applications/ppebl.app/Contents/MacOS/ppebl.

2.4 How to stop running a program

In order to improve performance, PEBL runs at the highest priority possible on your computer. This means that if it gets stuck somewhere, you may have difficulty terminating the process. We have added an ‘abort program’ shortcut key combination that will immediately terminate the program and report the location at which it became stuck in your code:
press <CTRL><SHIFT><ALT><\> simultaneously.

2.5 Command-line arguments

Some aspects of PEBL’s display can be controlled via command-line arguments. Some of these are platform specific, or their use depends on your exact hardware and software. The following guide to command-line arguments is adapted from the output produced by invoking PEBL with no arguments:

Usage: Invoke PEBL with the experiment script files (.pbl) and command-line arguments.

Examples:
pebl experiment.pbl -s sub1 --fullscreen --display 800x600
--driver dga
pebl experiment.pbl --driver xf86 --language es
pebl experiment.pbl -v 33 -v 2 --fullscreen --display 640x480

Command-Line Options

-v VALUE1 -v VALUE2
Invokes script and passes VALUE1 and VALUE2 (or any text immediately following a -v) to a list in the argument of the Start() function. This is useful for passing in conditions, subject numbers, randomization cues, and other entities that are easier to control from outside the script. Variables appear as strings, so numeric values need to be converted to be used as numbers.

-s VALUE
-s VALUE
Binds VALUE to the global variable gSubNum, which is set by default to 0.

--driver <drivername>
Sets the video driver, when there is more than one. In Linux SDL, options xf86, dga, svgalib (from console), it can also be controlled via environment variables. In fact, for SDL versions of PEBL simply set the
Chapter 2. Usage

SDL_VIDEO_DRIVER environment variable to the passed-in argument, without doing any checking, and without checking or returning it to its original state.

--display <widthxheight>

Controls the screen width and height (in pixels). Defaults to the current resolution of the screen. Unlike older versions of PEBL, after 0.12 any legal combination of width and height should work.

The screensize a PEBL script runs at depends on a number of things. If no --display size is given (e.g., when 'default' is chosen in the launcher), PEBL will try to determine the current screen size and use that, for both fullscreen and windowed mode. Otherwise, it will try to use the specified value.

However, these values are only a request. When the script starts, it sets the values of the global variables gVideoWidth and gVideoHeight based on either the specified values or the current screen size. These values can be changed in the script before the MakeWindow function is called, so that a script can require a particular screen size. Then, the window will be created with those dimensions, overriding any command-line parameters. For greatest flexibility, it is recommended that you do not hard-code screen size but rather make your test adapt to a large number of screen sizes.

Finally, if a screen size is selected that the video card cannot support (i.e., in fullscreen mode), gVideoWidth and gVideoHeight will be set to the legal screen size closest to the one you requested. PEBL should never crash because you have specified the wrong screen size, but it should rather use one it can support. The values of gVideoWidth and gVideoHeight will be changed by MakeWindow to whatever screen size it actually uses.

--depth

Controls the pixel depth, which also depends on your video card. Currently, depths of 2,8,15,16,24, and 32 are allowed on the command-line. There is no guarantee that you will get the specified bit depth, and bit depths such as 2 and 8 are likely never useful. Changing depths can, for some drivers and video cards, enable better performance or possibly better video synchrony. Defaults to 32.

--language

Allows user to specify a language code that can get tested for within a script to select proper translation. It sets a global variable gLanguage, and is “en” by default.

--windowed or --fullscreen Controls whether the script will run in a window or fullscreen. The screen resolution a PEBL script runs at depends on a number of things. See the --display option above for more details.
Chapter 2. Usage
Chapter 3

How to Write a PEBL Program

3.1 Basic PEBL Scripts

PEBL has a fairly straightforward and forgiving syntax, and implements most of its interesting functionality in a large object system and function library of over 125 functions. The library includes many functions specific to creating and presenting stimuli and collecting responses. Efforts, however successful, have been made to enable timing accuracy at millisecond-scale, and to make machine limitations easy to deal with.

Each PEBL program is stored in a text file. Currently, no special authoring environment is available. A program consists of one or more functions, and must have a function called \texttt{Start()}. Functions are defined with the following syntax:

\begin{verbatim}
define <function_name>(parameters)
{
  statement 1
  statement 2
  ....
  return value3
}
\end{verbatim}

The parameter list and the return value are optional. For the \texttt{Start(par)\{} function, \texttt{par} is normally bound to 0. However, if PEBL is invoked with \texttt{-v} command-line parameters, each value that follows a \texttt{-v} is added to a list contained in \texttt{`par'}, which can then be accessed within the program:
define Start(par)
{
    Print(First(par))
}

A simple PEBL program that actually runs follows:

define Start(par)
{
    Print("Hello")
}

Print() is a standard library function. If you run PEBL from a command-line, the text inside the Print function will be sent to the console. On Windows, it will appear in the file `stdout.txt' in the PEBL directory. Although other functions do not need a parameter argument, the Start() function does (case values are passed in from the command-line).

A number of sample PEBL programs can be found in the /demo subdirectory.

3.2 Case Sensitivity

PEBL uses case to specify an item's token type. This serves as an extra contextual cue to the programmer, so that the program reads more easily and communicates more clearly.

Function names must start with an uppercase letter, but are otherwise case-insensitive. Thus, if you name a function "DoTrial", you can call it later as "D0TRIAL" or "Dotrial" or even "DotRail". We recommend consistency, as it helps manage larger programs more easily.

Unlike function names, variable names must start with a lowercase letter; if this letter is a 'g', the variable is global. This enforces a consistent and readable style. After the first character, variable names are case-insensitive. Thus, the variable 'mytrial' is the same as 'myTrial'.

Currently, syntax keywords (like loop, if, define, etc.) must be lowercase, for technical reasons. We hope to eliminate this limitation in the future.

3.3 Syntax

PEBL has a simple and forgiving syntax, reminiscent of S+ (or R) and c. However, differences do exist.

Table 3.1 shows a number of keywords and symbols used in PEBL. These need not appear in lowercase in your program.

Note that the '=' symbol does not exist in PEBL. Unlike other languages, PEBL does not use it as an assignment operator. Instead, it uses '<-'. Because it is confusing for users to keep track of the various uses of the = and == symbols, we've eliminated the '=' symbol entirely. Programmers familiar with c will notice a resemblance between PEBL and c. Unlike c, in PEBL a semicolon is not necessary to finish a statement. A carriage return indicates a statement is
## Table 3.1: PEBL Symbols and Keywords

<table>
<thead>
<tr>
<th>Symbol/Keyword</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds two expressions together</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts one expression from another</td>
</tr>
<tr>
<td>/</td>
<td>Divides one expression by another</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies two expressions together</td>
</tr>
<tr>
<td>^</td>
<td>Raises one expression to the power of another</td>
</tr>
<tr>
<td>;</td>
<td>Finishes a statement, or starts a new statement on the same line (is not needed at end of line)</td>
</tr>
<tr>
<td>.</td>
<td>The property accessor. Allows properties to be accessed by name</td>
</tr>
<tr>
<td>&lt;-</td>
<td>The assignment operator</td>
</tr>
<tr>
<td>( )</td>
<td>Groups mathematical operations</td>
</tr>
<tr>
<td>{ }</td>
<td>Groups a series of statements</td>
</tr>
<tr>
<td>[ ]</td>
<td>Creates a list</td>
</tr>
<tr>
<td>#</td>
<td>Comment—ignore everything on the line that follows</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>=&gt;</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt; != !~ =~</td>
<td>Not equal to</td>
</tr>
<tr>
<td>and</td>
<td>Logical and</td>
</tr>
<tr>
<td>break</td>
<td>Breaks out of a loop prematurely</td>
</tr>
<tr>
<td>not</td>
<td>Logical not</td>
</tr>
<tr>
<td>or</td>
<td>Logical or</td>
</tr>
<tr>
<td>while</td>
<td>Traditional while loop</td>
</tr>
<tr>
<td>loop</td>
<td>Loops over elements in a list</td>
</tr>
<tr>
<td>if</td>
<td>Simple conditional test</td>
</tr>
<tr>
<td>if...else</td>
<td>Complex conditional test</td>
</tr>
<tr>
<td>if...elseif...else</td>
<td>Extended conditional chain</td>
</tr>
<tr>
<td>define</td>
<td>Defines a function</td>
</tr>
<tr>
<td>return</td>
<td>Allows a function to return a value</td>
</tr>
</tbody>
</table>
Chapter 3. How to Write a PEBL Program

3.4 Expressions

An expression is a set of operations that produces a result. In PEBL, every function is an expression, as is any single number. Expressions include:

- \(3 + 32\)
- \((324 / 324) - \log(32)\)
- \(\text{not lVariable}\)
- \(\text{Print}(32323)\)
- "String " + 33
- nsuho #this is legal if nsuho has been defined already.

Notice that "String " + 33 is a legal expression. It will produce another string: "String 33".

These are not expressions:

NSUHO #Not an expression
( 33 + 33 #Not an expression
444 / 3342 + #Not an expression

NSUHO is not a variable because it starts with a capital letter. The other lines are incomplete expressions. If the PEBL parser comes to the end of a line with an incomplete expression, it will automatically go to the next line:

\[
\text{Print(\text{"hello } + \\
\text{" world."}}
\]

This can result in bugs that are hard to diagnose:

\[
a <- 33 + 323 + \\
\text{Print(1331) }
\]

sets a to the string "3561331".

But if a carriage return occurs at a point where the line does make a valid expression, it will treat that line as a complete statement:

\[
a <- 33 + 323 \\
* 34245
\]
sets a equal to 356, but creates a syntax error on the next line.
Any expression can be used as the argument of a function, but a function may
not successfully operate when given bogus arguments.
If a string is defined across line breaks, the string definition will contain a
linebreak character, which will get printed in output text files and textboxes.

    text <- "this is a line
    and so is this"

If you desire a long body of text without linebreaks, you must define it piecemeal:

    text <- "This is a line " +
    "There is no line break before this line."

### 3.5 Variables

PEBL can store the results of expressions in named variables. Unlike many
programming languages, PEBL only has one type of variable: a “Variant”. This
variable type can hold strings, integers, floating-point numbers, lists, graphical
objects, and everything else PEBL uses to create an experiment. Unlike other
languages, a variable need not be declared before it can be used. If you try to
access a variable that has not yet been declared, PEBL will return a fatal error
that stipulates as such.

#### 3.5.1 Coercion/casting

Variants just hide the representational structure from the user. An actual string
resides within the variant that holds a string. A long integer resides within the
variant that holds an integer.

PEBL Variants are automatically coerced or cast to the most appropriate inner
format. For example, \(3232.2 + 33\) starts out as a floating point and an integer.
The sum is cast to a floating point number. Similarly, \"banana\" + 33 starts as
a string and an integer, but the combination is a string.

#### 3.5.2 Variable Naming

All variables must begin with a lowercase letter. Any sequence of numbers or
letters may follow that letter. If the variable begins with a lowercase ‘g’, it has
global scope; otherwise it has local scope.

#### 3.5.3 Variable Scope

As described above, variables can have either local or global scope. Any vari-
able with global scope is accessible from within any function in your program.
A variable with local scope is accessible only from within its own function. Dif-
ferent functions can have local variables with the same name. Generally, it is
a good idea to use local variables whenever possible, but using global variables
for graphical objects and other complex data types can be intuitive.
3.5.4 Copies and Assignment

Variables may contain various types of data, such as simple types like integers, floating-point ratio numbers, strings; and complex types like lists, windows, sounds, fonts, etc. A variable can be set to a new value, but by design, there are very few ways in which a complex object can be changed once it has been set. For example:

```plaintext
woof <- LoadSound("dog.wav")
meow <- LoadSound("cat.wav")
dog <- woof
```

Notice that `woof` and `dog` refer to the same sound object. Now you may:

```plaintext
PlayBackground(woof)
Wait(50)
Stop(dog)
```

which will stop the sound from playing. If instead you:

```plaintext
PlayBackground(woof)
Wait(50)
Stop(meow)
```

`woof` will play until it is complete or the program ends.

Images provide another example. Suppose you create and add an image to a window:

```plaintext
mWindow <- MakeWindow()
mImage <- MakeImage("test.bmp")
AddObject(mImage, mWindow)
Draw()
```

Now, suppose you create another variable and assign its value to `mImage`:

```plaintext
mImage2 <- mImage
Move(mImage2, 200, 300)
Draw()
```

Even though `mImage2` was never added to `mWindow`, `mImage` has moved: different variables now point to the same object. Note that this does not happen for simple (non-object) data types:

```plaintext
a <- 33
b <- a
a <- 55
Print(a + " " + b)
```

This produces the output:

```
55 33
```
Chapter 3. How to Write a PEBL Program

This may seem confusing at first, but the consistency pays off in time. The ‘<-’ assignment operator never changes the value of the data attached to a variable, it just changes what the variable points to. PEBL is functional in its handling of simple data types, so you can’t, for example, directly modify the contents of a string.

\begin{verbatim}
a <- "my string"       # assigns a string literal to a
b <- a                 # makes b refer to a's string literal
a <- "your string"     # re-assigns a to a new string literal
b <- a                 # makes b refer to a's new string literal
\end{verbatim}

3.5.5 Passing by Reference and by Value

The discussion in 3.5.4 on copying has implications for passing variables into functions. When a variable is passed into a function, PEBL makes a copy of that variable on which to operate. But, as discussed in 3.5.4, if the variable holds a complex data type (object or a list), the primary data structure allows for direct modification. This is practical: if you pass a window into a function, you do not want to make a copy of that window on which to operate. If the value is a string or a number, a copy of that value is made and passed into the function.

3.6 Functions

The true power of PEBL lies in its extensive library of functions that allow specific experiment-related tasks to be accomplished easily. For the sake of convenience, the library is divided into a number of subordinate libraries. This library structure is transparent to the user, who does not need to know where a function resides in order to use it. Chapter 5 includes a quick reference to functions; Chapter 9 includes a complete alphabetical reference.
3.7 A Simple Program

The previous sections provide everything you need to know to write a simple program. Here is an annotated program:

```plaintext
# Any line starting with a # is a comment. It gets ignored.

# Every program needs to define a function called Start()
# Start always needs a parameter
define Start(par)
{
    number <- 10  ##Assign a number to a variable
    hello <- "Hello World"  ##Assign a string to a variable
    ##Create a global variable (starts with little g)
gGlobalText <- "Global Text"
    ##Call a user-defined function (defined below).
    value <- PrintIt(hello, number)
    ##It returned a value
    # Call a built-in function
    Print("Goodbye. " + value)
}

## Define a function with two variables.
define PrintIt(text, number)
{
    # Seed RNG with the current time.
    RandomizeTimer()
    # Generate a random number between 1 and number
    i <- RandomDiscrete(number)  # this is a built-in function
    ## Create a counter variable
    j <- 0
    ## Keep sampling until we get the number we chose.
    while(i != number)
    {
        Print(text + " "+ i + gGlobalText)
        i <- RandomDiscrete(number)
        j <- j + 1
    }
    return(j)  # return the counter variable.
}
```

More sample programs can be found in the demo/ and experiments/ directories of the PEBL source tree.
Chapter 4

Overview of Object Subsystems

In PEBL, complex objects are stored and automatically self-managed. These objects include lists, graphical display widgets like images and text displays, fonts, colors, audio files, and input or output files. Objects are created and modified with special functions, but many of their properties available directly for access and modification with a `variable.property` syntax. For example, the position of a textbox is controlled by `.X` and `.Y` properties, and can also be changed with the `Move()` function. To move the label `lab`, which is located at 100,100, to 150,100, you can either do `Move(lab,150,100)` or `lab.X <- 150`. The available properties and accessor function are listed in the descriptions of their relevant objects below.

4.1 Lists

Lists are incredibly useful and flexible storage structures that play an important role in PEBL. A list is simply a series of variables. It is the equivalent to a vector, array, or other similar data structure in many other programming languages. Creating and accessing elements of lists can be accomplished in a number of ways. If you have a set of values you want to create a list from, you simply need to put them inside square brackets, separated by commas:

```r
mylist <- [1,2,3,4,5,6,7,8,9]
```

Many functions related to experimental design return lists already created. Two simple functions are `Repeat` and `Sequence`:

```r
list1 <- Repeat(0,10)  ##ten zeroes
list2 <- Sequence(0,20,2)  ##numbers 0 to 20 step 2
```

Accessing list items can be done in a number of ways. The simplest is using the `Nth()` function. For a slightly more complex example, suppose you want to
print out every item in a list. Looping through, accessing, and printing all the
items of a list using this approach:

```r
list <- Sequence(1, 9, 1)
len <- Length(list)
i <- 1
while (i <= len)
{
  item <- Nth(list, i)
  Print(item)
  i <- i + 1
}
```

Note that prior to PEBL 0.12, using Nth to access list items was inefficient.
Since PEBL 0.13, you can use Nth to access list items in amortized constant
time! But nevertheless, the above method of looping is verbose and error-prone.
There is an alternative. Items from lists can be iterated over using the ‘loop’
command:

```r
list <- Sequence(1, 9, 1)
loop(item, list)
{
  Print(item)
}
```

These two code blocks produce identical output, but in the former block, each
item of the list must be found on each iteration, but in the latter block, a list
item is bound directly to ‘item’ on each iteration. There is no appreciable
difference in the efficiency of these two methods, but the second is simpler and
in many cases easier to use, and avoids some errors (like forgetting to increment
i).

### 4.1.1 Growing Lists

Oftentimes, you want to create a list one element at a time. For example,
you may have a sampling scheme for stimuli and need to pick each consecutive
randomly, or you want to record response times or accuracies one trial at a time.
There are two ways you can do this. If you know how long your list will be,
you can create a list with as many elements as you need, and then alter each
element one at a time.

```r
##I need ten items
items <- Repeat(0, 10)
i <- 1
```
```plaintext
Chapter 4. Overview of Object Subsystems

while(i <= Length(items))
{
  SetElement(items,i,Random())
}

Oftentimes, however, this is difficult because you do not know how long the list should be at the beginning. The Append() function is able to add an item to the end of a list, and you can use that to 'grow' a list one item at a time:

#I need ten items

items <- []
i <- 1
while(i <= 10)
{
  items <- Append(items,Random())
}

This idiom is used in many places in PEBL test batteries. However, it can be inefficient as the length of the list grows. This is because on each iteration, a new list is created that is 1 element longer than the previous list (and each element is copied to the new list). For small lists, even ones hundred of items long, this overhead is pretty small and you hardly notice. But as a list gets thousands of items long, this can start to slow things down, especially if you are doing something complex between each trial. As of PEBL 0.13, we support another function called PushOnEnd():

items <- []
i <- 1
while(i <= 10)
{
  PushOnEnd(items,Random())
}

PushOnEnd will alter items directly, and do so in a very efficient way. Notice that you don't need to copy the new list and overwrite itself. However, for ease of use, PushOnEnd() returns the current copy of the list, and so you can often use it as a drop-in replacement for Append (in cases where you are throwing away the original list). In tests, this method appears to be only 5-10% less efficient than using PushOnEnd alone, and so it should hardly be noticed.

items <- []
i <- 1
while(i <= 10)
{
  items <- PushOnEnd(items,Random())
}
```

19
A caveat when using lists: Some functions operate on lists to produce new lists (sub-lists, re-ordered lists, etc.). When the lists contain simple data types (numbers, strings, etc.), entirely new data structures are created. But when the data structures are complex (windows, sounds, images, etc.), the objects are not copied. Only new pointers to the original objects are created. So if you change the original object, you may end up accidentally changing the new object. Although that is relatively difficult, because PEBL allows only limited modification of existing data structures, it is still possible. This is a special case of the copy/assignment issue discussed in Section 3.5.4: Copies and Assignment.

4.1.2 Recursion on lists

Many mathematical functions that take a single argument can be applied either to a number or a list of numbers. When applied to an entire list, it will return the function applied to each element of that list. For example, Ln(1) return 0, but Ln([1,1,1]) returns [0,0,0].

A list of functions that support this include:
- Log10
- Log2
- Ln
- Exp
- Sqrt
- Tan
- Sin
- Cos
- ATan
- ASin
- ACos
- DegToRad
- RadToDeg
- Round
- Floor
- Ceiling
- AbsFloor
- Sign
- Abs

In addition, a number of math functions that take two arguments will apply themselves recursively to the first argument should it be a list. For example, LogN([1,1,1,5]) will return [0,0,0]. Functions that support this include:
- LogN
- Pow
- NthRoot
4.2 Fonts

PEBL uses truetype fonts for the display of text in labels and other text widgets. In addition to the filename, font objects have the following properties: style (i.e., normal, bold, italic, underline), size (in points), foreground color, background color, and whether it should be rendered anti-aliased.

We distribute a series of high-quality freely available and redistributable fonts, including the DejaVu series, freefont series, and a few others. These include the typeface/ files shown below 4.1:

These should always be available for use in experiments. The fonts.pbl script in the demo/ directory will display what symbols from each of these fonts looks like.

To use, you need only specify the font name in the MakeFont() function:

```r
colorRed <- MakeColor("red")
colorGrey <- MakeColor("grey")
myFont <- MakeFont("VeraMono.ttf",0,22,colorRed,colorGrey,1)
```

This code makes a red 22-point anti-aliased font on a grey background. Other fonts may be used by specifying their absolute pathname or copying them to the working directory and using them.

Accessible font properties:

- font.FILENAME
- font.BOLD
- font.UNDERLINE
- font.ITALIC
- font.SIZE
- font.FGCOLOR
- font.BGCOLOR
- font.ANTIALIASED

Having the right fonts is important for translating PEBL scripts into new languages. Previously, this was challenging because the default font used in many scripts was Vera, and Vera has poor support for international characters. As of PEBL 0.11, a few things have changed to make international character support easier:

- Three new fonts that support international characters much better ("DejaVuSans.ttf", "DejaVuSansMono.ttf", and "DejaVuSerif.ttf") are now included and available.

- Three new global variables are set on initiation: gPEBLBaseFont, gPEBLBaseFontMono, and gPEBLBaseFontSerif, which are set by default to these three font names.

- Helper functions and battery tests are all updated to use these values to set up fonts.
### Table 4.1: Typeface/Files Available in PEBL

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeSans.ttf</td>
<td>FreeFont Fonts</td>
</tr>
<tr>
<td>FreeSansBold.ttf</td>
<td>Simple Clean sans serif font</td>
</tr>
<tr>
<td>FreeSansOblique.ttf</td>
<td></td>
</tr>
<tr>
<td>FreeSansBoldOblique.ttf</td>
<td></td>
</tr>
<tr>
<td>FreeMono.ttf</td>
<td>Courier-like fontface</td>
</tr>
<tr>
<td>FreeMonoBold.ttf</td>
<td></td>
</tr>
<tr>
<td>FreeMonoOblique.ttf</td>
<td></td>
</tr>
<tr>
<td>FreeSerif.ttf</td>
<td></td>
</tr>
<tr>
<td>FreeSerifBold.ttf</td>
<td></td>
</tr>
<tr>
<td>FreeSerifOblique.ttf</td>
<td></td>
</tr>
<tr>
<td>FreeSerifBoldOblique.ttf</td>
<td></td>
</tr>
<tr>
<td>Caliban.ttf</td>
<td>Fontforge Fonts</td>
</tr>
<tr>
<td>Helvetica-style</td>
<td></td>
</tr>
<tr>
<td>CaslonRoman.ttf</td>
<td>Quirky Roman Font series</td>
</tr>
<tr>
<td>CaslonBold.ttf</td>
<td></td>
</tr>
<tr>
<td>CaslonItalic.ttf</td>
<td></td>
</tr>
<tr>
<td>Caslon-Black.ttf</td>
<td></td>
</tr>
<tr>
<td>Humanistic.ttf</td>
<td>Sharp, refined fontface</td>
</tr>
<tr>
<td>DoulosSILR.ttf</td>
<td>SIL Fonts</td>
</tr>
<tr>
<td>Comprehensive font with roman and cyrillic glyphs</td>
<td></td>
</tr>
<tr>
<td>GenR102.ttf</td>
<td>Includes many latin alphabet letters</td>
</tr>
<tr>
<td>GenI102.ttf</td>
<td></td>
</tr>
<tr>
<td>CharisSILR.ttf</td>
<td>Like doulos, optimized for printing</td>
</tr>
<tr>
<td>CharisSILB.ttf</td>
<td></td>
</tr>
<tr>
<td>CharisSILI.ttf</td>
<td></td>
</tr>
<tr>
<td>CharisSILBI.ttf</td>
<td></td>
</tr>
<tr>
<td>Stimulasia.ttf</td>
<td>PEBL Fonts</td>
</tr>
<tr>
<td>A small set of arrow/boxes</td>
<td></td>
</tr>
<tr>
<td>Vera.ttf</td>
<td>Bitstream Vera Series (Deprecated in favor of DejaVu)</td>
</tr>
<tr>
<td>Sans serif Roman-style base font</td>
<td></td>
</tr>
<tr>
<td>VeraMono.ttf</td>
<td>Sans serif Roman-style mono-spaced base font</td>
</tr>
<tr>
<td>VeraSe.ttf</td>
<td>Serif Roman-style base font (similar to times)</td>
</tr>
<tr>
<td>VeraBd.ttf</td>
<td>Bold Vera</td>
</tr>
<tr>
<td>VeraIt.ttf</td>
<td>Italic Vera</td>
</tr>
<tr>
<td>VeraBI.ttf</td>
<td>Bold Italic Vera</td>
</tr>
<tr>
<td>VeraMoBd.ttf</td>
<td>Bold Vera Mono</td>
</tr>
<tr>
<td>VeraMoIt.ttf</td>
<td>Italic Vera Mono</td>
</tr>
<tr>
<td>VeraMoBI.ttf</td>
<td>Bold Italic Vera Mono</td>
</tr>
<tr>
<td>VeraSeBd.ttf</td>
<td>Bold Serif Vera</td>
</tr>
<tr>
<td>DejaVuSerif.ttf</td>
<td>DejaVu Series (Version of Vera with international characters)</td>
</tr>
<tr>
<td>Serif Roman-style base font</td>
<td></td>
</tr>
<tr>
<td>DejaVuSans.ttf</td>
<td>Serif Roman-style base font</td>
</tr>
<tr>
<td>DejaVuSansMono.ttf</td>
<td>Sans serif Roman-style mono-spaced base font</td>
</tr>
<tr>
<td>wqy-zenhei.ttc</td>
<td>2CJK Fonts</td>
</tr>
<tr>
<td>All-purpose font with support for Chinese, Korean and Japanese</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4. Overview of Object Subsystems

So now, many international characters will be handled by default. For character sets that aren't handled by DejaVu, simply needs to change `gPEBLBaseFont` to name a font that can handle your characters (and include that font in the program directory), and everything should work out fine.

### 4.3 Colors

Colors are PEBL objects. A color can be created by specifying its name using the `MakeColor()` function, or by specifying its RGB values using the `MakeColorRGB()` function. A list of colors and their respective RGB values can be found in the `Colors.txt` file in the documentation directory, or in the final chapter of the manual. There are nearly 800 from which to choose, so you can create just about anything you can imagine.

Accessible color properties:

- `color.RED`
- `color.GREEN`
- `color.BLUE`
- `color.ALPHA`

### 4.4 Windows

To run an experiment, you usually need to create a window in which to display stimuli. This is done with the `MakeWindow()` function. `MakeWindow()` will create a grey window by default, or you can specify a color. Currently, an experiment can have only one window.

### 4.5 Graphical Widgets

Graphical “widgets” are the building blocks of experimental stimuli. Currently, four widgets are available: images, labels, canvases, and textboxes. More complicated widgets are in progress or planned. There are also a number of shapes that in some ways behave like widgets, but are technically not.

To be used, a widget must be created and added to a parent window, and then the parent window must be drawn. You can hide widgets with the `Hide()` function, and show them with the `Show()` function; however, this affects only the visibility of the widget: it is still present and consuming memory. Widgets can be moved around on the parent window using the `Move()` function. `Move()` moves the center of an image or label to the specified pixel, counting from the upper-left corner of the screen. `Move()` moves the upper left-hand corner of textboxes. For the sake of convenience, the `MoveCorner` function is available, which will move an image or label by its upper left-hand corner.

You should remove widgets from their parent window when you are finished using them.

All widgets have several properties available for controlling their behavior.

```
widget.name
```
4.6 Images

PEBL can read numerous image types, courtesy of the SDL_image library. Use the `MakeImage()` function to read an image into an image object. As images are often used as stimuli, `Move()` centers the image on the specified point. To move by the upper-left hand corner, use the PEBL-defined `MoveCorner()` function:

```plaintext
define MoveCorner(object, x, y)
{
    size <- GetSize(object)
    centerX <- x + First(size)/2
    centerY <- y + Last(size)/2
    Move(object, centerX, centerY)
}
```

Images have all the properties available for widgets, but the width and height can only be read, and not set. Width and height are controlled by the dimensions of the image file.

4.7 Canvases

A canvas is a blank rectangle, sort of like an 'imageless' image. As with an image, `Move()` centers the image on the specified point. A canvas appears similar to a `Rectangle()` shape, but differs in some important ways. First, a Canvas has a piece of video memory associated with it—shapes do not. This means that other objects can be added to a canvas, just as it can be added to a window. If you move the canvas around, the attached objects will move with the canvas.

Second, individual pixels of a canvas can be set, using the `SetPoint()` function. `SetPoint()` works on images too, but not on text. This is because a `Draw()` command re-renders text, and so will wipe out any pixel damage you have done. This can be useful for making special-purpose drawing functions to create stimuli, especially noise distributions.

Finally, a canvas can be drawn on with another object. In fact, you can add another image as a brush. Add an image to a canvas, and anytime you call `Draw()` on the canvas (rather than without an argument), the image gets imprinted on the canvas. This will remain until you call `ResetCanvas()`.

For example:
Chapter 4. Overview of Object Subsystems

```
tb <- MakeCanvas(600,400,d)
AddObject(tb,win)
##add the image to canvas, not win
pebl <- MakeImage("pebl.png")
AddObject(pebl,tb)
##Nothing will appear on the screen in these intermediate draws
Move(pebl,100,100)
Draw(pebl)
Move(pebl,200,100)
Draw(pebl)
Move(pebl,200,200)
Draw(pebl)
Move(pebl,100,200)
Draw(pebl)

Draw() ##Now, we will see the canvas with 4 pebl images on it.
```

The draw-on trick can be used to add noise to a text stimulus. Make a label and add it to a canvas, use Draw() on the label, then hide the label, and add noise to the canvas by using SetPoint(). Anything drawn on the canvas won’t get reset until the ResetCanvas() function is called.

Images have all the properties available for widgets. Size cannot be updated once the canvas is created.

Note that the background color can have an alpha value. If you use an alpha value of 0, the background will be invisible.

4.8 Shapes

PEBL allows you to define a number of shape objects that can be added to another widget. A demonstration script exercising these shapes is found in demo/shapes.pbl.

The following is a list of shape and their properties.

4.8.1 Circle

Description: A standard circle. Move commands move the center of the circle to the specified location.

Command: Circle(<x>,<y>,<r>,<color>,<filled>)

Properties: .name
.filled = 0,1 (whether it is filled)
.color (color)
.x (x position of center)
.y (y position of center)
4.8.2 Ellipse

*Description:* An ellipse, with height and width differing. Cannot be pointed in an arbitrary direction. Move commands move the center of the shape to the specified location.

*Command:* `Ellipse(<x>,<y>,<rx>,<ry>,<color>,<filled>)`

*Properties:*  
- `.name`  
- `.filled = 0,1` (whether it is filled)  
- `.color` (color)  
- `.x` (x position of center)  
- `.y` (y position of center)  
- `.height` (read-only height)  
- `.width` (read-only width)  
- `.rx` (x radius)  
- `.ry` (y radius)

4.8.3 Square

*Description:* A square. Move commands move the center of the shape to the specified location.

*Command:* `Square(<x>,<y>,<size>,<color>,<filled>)`

*Properties:*  
- `.name`  
- `.filled = 0,1` (whether it is filled)  
- `.color` (color)  
- `.x` (x position of center)  
- `.y` (y position of center)  
- `.height` (read-only height)  
- `.width` (read-only width)  
- `.dx`, `.dy`, `.size` (Length of side)

4.8.4 Rectangle

*Description:* A Rectangle. Move commands move the center of the rectangle to the specified location.

*Command:* `Rectangle(<x>,<y>,<dx>,<dy>,<color>,<filled>)`
Chapter 4. Overview of Object Subsystems

Properties: .name
.filled = 0,1 (whether it is filled)
.color (color)
.x (x position of center)
.y (y position of center)
.height (read-only height)
.width (read-only width)
.dx, (width) .dy, (height)

4.8.5 Line

Description: A Line. Move commands move the center of the line to the specified location.
Command: Line( <x>, <y>,<dx>,<dy>,<color>)

Properties: .color (color)
.x (x position of start)
.y (y position of start)
.width, (x length)
.height, (y length)

4.8.6 Polygon

Description: An arbitrary polygon.
Command: Polygon( <x>, <y>,<xpoints>,<ypoints>,<color>,<filled>)

Properties: .name
.color (color)
.x (x position of start)
.y (y position of start)

4.8.7 Bezier

Description: An arbitrary bezier curve.
Command: Bezier( <x>, <y>,<xpoints>,<ypoints>,<steps>,<color>)

Properties: .name
.color (color)
.x (x position of start)
.y (y position of start)
Chapter 4. Overview of Object Subsystems

4.9 Text Labels

You can create a text label object with the MakeLabel() function, which requires specifying a font, and the foreground and background colors. Labels are only a single line of text. Like images, when you move them, they center on the specified point.

The text inside a label can be extracted with GetText() and set with SetText(). When you change a text object, it will not appear until the next time you call a Draw() function.

Text labels have all the regular widget properties, plus:

- label.TEXT
- label.FONT

The .HEIGHT and .WIDTH accessible, but cannot be changed because they are controlled by the text and the font size.

4.10 Text Boxes

A text box is a graphical widget that contains a body of text. Text automatically wraps when it is too long to fit on a single line. Like labels, the text inside a Text Box can be extracted with GetText() and set with SetText(). When a text object is changed, it rerenders immediately, but does not appear until the next time a Draw() function is called.

Text box properties:

- textbox.EDITABLE
- textbox.CURSORPOS
- textbox.DIRECTION
- textbox.LINEHEIGHT
- textbox.LINEWRAP

4.11 User-Editable Text Boxes

Text box editing can be performed using the GetInput(<textbox>,<escape-key>) function. This returns the text that is present in the box when the participant hits the key associated with <escape-key>. <escape-key> is just a text-based code that describes the keypress that should be checked for exit. Typical escape-key options include:

- "<return>"
- "<esc>"
- "<backspace>"
- "<kp_enter>"
- "A"

See the Keyboard Entry section below for a more complete list.
Translation from string to keyboard input is still crude, and is handled in
src/utility/PEBLUtility.cpp:TranslateString
Chapter 4. Overview of Object Subsystems

4.12 Audio

Currently, audio output is very primitive, and there are no facilities for recording or analyzing audio input. Audio .wav files can be loaded with the LoadSound() function, which returns an audio stream object that can be played with either the PlayForeground() or PlayBackground() functions. The PlayForeground() function returns once the sound is finished playing; PlayBackground() returns immediately and the sound plays in a separate thread. When using PlayBackground, playing can be stopped using the Stop() function. If another PlayForeground() or PlayBackground() is then used, the initial sound will immediately terminate and the new file will play. Currently, PEBL can only play one sound at a time.

4.13 Movie Files

If compiled to support them, PEBL can read numerous video and audio media files waave library and ffmpeg. Use the LoadMovie() function to read a movie file into a movie object. The Move() function moves the upper left corner of the movie to the specified point. An audio file can be similarly loaded using the LoadAudioFile function.

Movie playback is done via a handler placed in the event loop. This handler is placed there with the StartPlayback function. Then, when the event loop runs, the movie will get updated in proper time sequence. The event loop is used for most WaitFor type events. This allows you to play a movie and wait for a response at the same time. Alternately, a complete movie file can be played in full (with no possibility for stopping early) using the PlayMovie() function. Movies have a number of properties that can be set to change playback or determine aspects of the movie. These are all accessible via .property syntax, and can be printed by the PrintProperties function. Properties include:

- DURATION: time in ms
- FILENAME: filename
- HEIGHT: pixels high
- NAME: <MOVIE>
- PLAYBACKPOSITION: where playback is
- ROTATION: Inherited; will not work
- VISIBLE: whether hidden or visible
- VOLUME: volume on a logarithmic scale—can go from 0 to +infinity
- WIDTH: screen width in pixels
- X: upper left corner x
Chapter 4. Overview of Object Subsystems

- Y: upper left corner y
- ZOOMX: scaling; not used (just set width)
- ZOOMY: scaling; not used (just set height)

### 4.14 Custom objects

Along with the built-in objects, PEBL lets you create your own object with properties that can be added, changed, and accessed using the .property notation. With appropriate use of the CallFunction command, you can also specify function handlers for functions such as Move(), Draw(), Inside(), or whatever you want. The object system in PEBL is fairly (and intentionally) primitive, without things you might expect from full-fledged object-oriented languages (i.e., accessor functions, inheritance, methods, constructors, etc.). Nevertheless, it can be very useful for encapsulating a lot of information about a computing object, and is used heavily in the GUI objects found in the launcher and other PEBL tools.

Use MakeCustomObject(name) to create a custom object. Then, a property can be added by assigning obj.name. For example, suppose you want an object to represent the x,y location of a point.

```plaintext
p1 <- MakeCustomObject("point")
p1.x <- 100
p1.y <- 100
```

Now, if you want to use access the x and y properties, do:

```plaintext
Print("position is:" p1.x + "," + p1.y)
```

An object can take a function name as a property. For example:

```plaintext
p1.inside <- "InsidePoint"
```

With the function InsidePoint defined as:

```plaintext
define InsidePoint(x,y,p)
{
  return (x==p.x and y==p.y)
}
```

If you had a bunch of objects, you could define the .inside property of each differently. Then, later, you could define InsideObject to check any of them:

```plaintext
define InsideObject(x,y,p)
{
  CallFunction(p.inside, [x,y,p])
}
```
Chapter 4. Overview of Object Subsystems

4.15 Keyboard Entry

PEBL can examine the state of the keyboard, and wait for various keyboard events to happen. Functions such as \texttt{WaitForKeyDown()}, \texttt{WaitForAnyKeyDown()}, etc., allow you to collect responses from subjects. Most keys are specified by their letter name; others have special names:

\begin{itemize}
  \item \	exttt{<left>}
  \item \	exttt{<up>}
  \item \	exttt{<down>}
  \item \	exttt{<right>}
  \item \	exttt{<enter>}
  \item \	exttt{<return>}
  \item \	exttt{<esc>}
  \item \	exttt{<backspace>}
  \item \	exttt{<kp_0>} through \texttt{<kp_9>}, as well as \texttt{<kp_period>}, \texttt{<kp_divide>}, \texttt{<kp_multiply>}, \texttt{<kp_minus>}, \texttt{<kp_plus>}, \texttt{<kp_equals>}, \texttt{<kp_enter>}
  \item \	exttt{<insert>}, \texttt{<delete>}, \texttt{<home>}, \texttt{<end>}, \texttt{<pageup>}, \texttt{<pagedown>}
\end{itemize}

For other special keys, check \texttt{<F1> through <F15>}, function keys.

Also, the traditional "modifier" keys can serve as normal keys:

\begin{itemize}
  \item \texttt{lshift}, \texttt{rshift}
  \item \texttt{numlock}, \texttt{capslock}, \texttt{scrolllock}, \texttt{rctrl}, \texttt{lctrl}, \texttt{ralt}, \texttt{lalt}, \texttt{meta}, \texttt{meta}
\end{itemize}

4.16 Joystick Input

PEBL supports input with a joystick. In order to use a joystick, you first need to poll the computer to determine whether a joystick is attached, and create a joystick object. The file joystickeystest.pbl in the demo directory creates a simple visual depiction of a fairly standard gamepad.

A joystick will have up to four types of inputs on it: buttons, axes, hats, and balls. But different joysticks are different, and so you may need to do some checking and testing for your particular setup. PEBL currently does not support force-feedback or rumble functions available on some joysticks.

Axes:

Each axis takes on a value between 1 and 32768. For a normal hand-grasp joystick, the first two axes will be determined by the relative x and y positions of the joystick. Gamepads often have triggers that are additional axis, or sometimes there are throttles (or gas/brake pedals in driving devices) that are mapped to axes. Find out how many axes exist with \texttt{GetNumJoystickAxes()}. Get the state of a particular axis with \texttt{GetJoystickAxisState()}. 

32
Chapter 4. Overview of Object Subsystems

Hats:
Hats are the little 8-way buttons that control direction on many game pads. They are sort of a digital axis, because each state is absolute. The entire hat state takes on a single integer number between 0 and 15. It is binary coded to specify whether each of the four major axes buttons are depressed:

- left: 8
- bottom: 4
- right: 2
- top: 1

The mechanics of the hat allows two buttons to be pressed simultaneously, indicating, for example, southeast or northwest. An example of how to extract the bitwise button states is found in the joysticktest.pbl file.

Buttons:
Usually, the state of each of the buttons on the joystick can be identified. Button state is coded so that 0=unpressed, 1=pressed. There can easily be a dozen or more buttons on a joystick, enabling some pretty elaborate response modes for experiments.

Balls:
Balls are very rare; you may have seen them in old-style arcade games like Arkenoid. No consumer joysticks available today appear to have balls that operate this way, and they have not been tested in PEBL. If you want to support trackballs, there are plenty of trackball mice that work as normal joystick controllers.

A number of functions are available for creating a joystick object and polling the joystick's current state:
Summary of joystick functions:

GetNumJoysticks()
OpenJoystick()
GetNumJoystickAxes()
GetNumJoystickBalls()
GetNumJoystickButtons()
GetNumJoystickHats()
GetJoystickAxisState()
GetJoystickHatState()
GetJoystickButtonState()

Currently, the joystick state is not integrated into PEBL's event loop. Consequently, there are no functions such as WaitForJoystickButton(), and no way
to create or monitor events. To use the joystick, you need to monitor the state of the device manually, and create a polling loop yourself, like:

```r
##This will keep looping until you press the first button
js <- OpenJoystick(1)  ##open the first joystick connected to the system
gCont <- 1
while (gCont)
{
  state <- GetJoystickAxisState(js,1)
  Print(state)
  ##Do something with the axis1 here

  gCont <- GetJoystickButtonState(1)
}
```

The file demo/ JoystickTest.pbl uses most of the available joystick functions to display a virtual gamepad on the screen as it captures input.

### 4.17 Files

Files are objects that can be read from or written to using several PEBL functions. To use a file object, create one using one of the functions listed below. Each function returns a file object:

- `FileOpenRead()`
- `FileOpenWrite()`
- `FileOpenOverwrite()`
- `FileOpenAppend()`

For example, you can use the command:

```r
myfile <- FileOpenRead("stimuli.txt")
```

to create `myfile`, a readable file stream.

`FileOpenWrite` is made with a safety backup. It will never overwrite an existing file; instead it will create a new file name by appending a number (i.e. 1) to the end of the base file name. The new filename will be saved as in the .filename property of the resulting file. The function `FileOpenOverwrite()` will overwrite any existing files, and should not be used for data files when you have the chance for a collision in a subject code.

The function `GetNewDataFile()` offers another all-in-one path for creating a data file. It will take a base name and a subject identifier, a file extension, and a header. If the file has not previously been created, it will create the new file and add a header row. If it has been created before, it will ask you whether to append to the current data file (in which case it won’t add a header row), or select a new data file. Data files are created in separated subdirectories (one per participant) within the data directory of the experiment.
Other Functions described below allow filestream to be written to or read from. When you are finished, you can close a filestream using the `FileClose()` function.

A number of related functions have been created to help make reading and writing to files easier. For example, the following functions enable reading an entire file into either a string variant, a list (with one list item per row), or a table:

- `ReadCSV()`
- `FileReadCharacter()`
- `FileReadLine()`
- `FileReadWord()`
- `FileReadTable()`
- `FileReadText()`
- `FileReadList()`
- `FileExists()`

### 4.18 Network Connections

PEBL has limited ability to open and communicate via TCP/IP connections, either some other system (e.g., for synchronizing with an e.g. or eyetracking computer), or another computer running PEBL (e.g., to create multi-subject game theory experiments or to have an experimenter controlling the task from another computer.)

#### 4.18.1 TCP/IP Overview

TCP/IP is a protocol by which computers can talk to one another. It is fairly barebones, and PEBL tries to hide much of its complexity. The information you send from one computer to another is guaranteed to arrive in the correct order, at the potential cost of serious delays, especially if the computers are on different networks or in different locations. Furthermore, connecting PEBL to another computer in this way is a potential security risk. However, the ability to transfer information between computers opens up huge potential for the types of experiments that can be constructed.

#### 4.18.2 Addresses and Ports

To do this, you first must open a network object to communicate with another computer. To do this, you must know (1) the IP number (like 127.0.0.1) or hostname (like myname.myschool.edu) of the computer you want to connect to, and (2) the port you want to connect on. You can even use the protocol to connect to another program running on your own computer, by specifying an IP address of 127.0.0.1, or the hostname “localhost”. A port is a number-usually 2 to 5 digits, specifying a type of service on your computer. Many ports are frequently used for specific types of communication, but you can use any port you wish to communicate, as long as both computers know this port. Most
ports on your computer should be blocked by default, so you may need to turn off your firewall or allow your chosen port to pass through the security or you may have trouble communicating.

To allow two PEBL programs to communicate, you need to decide that one computer is the “server” and the other is the “client”. On the server, you execute the function `WaitForNetworkConnection(port)`, which listens on the specified port until the client tries to connect. After the server is started, the client calls `ConnectToHost(hostname, port)` or `AcceptNetworkConnection(port)`, depending upon whether you are using the hostname or ip address. Typically, ip numbers are specified by four three-digit numbers separated by dots, like 192.168.0.1. This actually represents a 4-byte integer, and this 4-byte integer is what `ConnectToIP()` expects. To create that integer, use the function `ConvertIPString(ipnum)`, which accepts an IP address specified in a string.

So, you can use:
```
net <- ConnectToIP(ConvertIPString("127.0.0.1"), 1234)
```
to create a connection to another program listening on port 1234 on your own computer. These functions all return a network object (e.g., `net`) that must be used in later communication.

### 4.18.3 Sending and Receiving Data

Once connected, the distinction between client and server essentially disappears. However, to communicate, one computer must send data with the `SendData(net, data)`, and the other must receive the data, using the `GetData(net, size)` function. PEBL can only send text strings, and you must know the length of the message you want to receive. More complex communication can be done by creating a set of PEBL functions that encapsulate messages into text strings with templated headers that specify the message length. Then, to receive a message, you first read the fixed-length header, determine how much more data needs to be read, then read in the rest of the data.

### 4.18.4 Closing networks

If you are using a network connection to synchronize timing of two computers, you probably want to close the network connection with `CloseNetworkConnection(net)` after you have synchronized, to avoid any extra overhead.

A simple example of an experiment that uses TCP/IP to communicate is the NIM game in demo/nim.pbl.

### 4.19 Parallel Port

Starting with Version 0.12, PEBL can send and receive information via a standard parallel (printer) port. These don’t appear on many computers anymore, but you can still get them, and they are still important ways to interface with hardware devices such as EEG and MRI machines and homebrew button boxes.
Currently, parallel port access is fairly limited to setting and getting the state of the 8 data bits. Parallel ports have a number of bits you can play with, but currently PEBL only supports the basic 8 data bits. Basically, you can set the state of the bits or read the state of the bits, which can either control things like LEDs, or be impacted by making connections between the ground and the data bit.

If you have a parallel port, it is mapped to one of three ports: LPT1, LPT2, or LPTX. To initialize access to a port, you must call OpenPPort with the name of your port:

```r
port <- OpenPPort("LPT1")
```

Parallel ports have two modes, input and output. To read data in, it needs to be in input mode; to change the state of the bits, it needs to be in output mode. Set the state with `SetPPortMode(port,"<input>")` or `SetPPortMode(port,"<output>")`.

To access the state of a port, use `GetPPortState(port)`. It will return a string of "|" separated 1s and 0s, which specify the state of each of the 8 bits. To set the state of the port, use `SetPPortState(port,state)`. state should be a list of 8 0s or 1s:

```r
SetPPortState(port, c(0,0,0,0,0,0,0,1))
```

The internal c++ parallel port classes have substantially more flexibility, and can be adapted to do more complex access of parallel ports.

### 4.20 Serial Port

- OPENCOMPORT
- COMPORTSENDBYTE
- COMPORTGETBYTE

### 4.21 The Event Loop

To assist in testing for multiple input events simultaneously, PEBL implements an event loop that will quickly scan multiple conditions and execute proper results whenever any one condition is met. The event loop works by maintaining a list of triggers that can be satisfied by various conditions. The conditions typically specify a device or other data source to examine, such as the timer. On each cycle of the loop, all events are examined, and when any of them are satisfied, either a specified function will be executed, or the event loop will exit. Most of the timing and input functions use the event loop behind the scenes.

As of Version 0.12, simple means to program the event loop are available. Three functions include:

- `RegisterEvent()`. This allows you to specify a condition and a function name which executes whenever the condition is true.
Chapter 4. Overview of Object Subsystems

- `StartEventLoop()`. This starts the event loop, with all available events.
- `ClearEventLoop()`. This clears out the event loop so other events can be used.

Note that because other functions, such as `Wait()`, use the event loop, you can pre-load extra events and start the event loop with one of these functions. These are used in a number of test battery tasks. However, their use is currently somewhat experimental, and their names and arguments may change in the future, and so we will not provide a detailed description of their use here.

### 4.22 Parameter Setting

PEBL offers an interface to set a large set of experimental parameters from a text file. Furthermore, the PEBL launcher allows you to edit and save new parameter sets. This allows you to create common versions of a test that you call, without editing the PEBL script.

The PEBL parameter system is based on a custom object created with the `CreateParameters()` function. To use this, it requires you to set default parameters (in case the parameter file is not found or damaged). Create default parameters as a nested list containing property-value pairs. For example:

```r
parameters <- ["length", 10],
               ["trialsperblock", 15],
               ["numblocks", 3]
```

If you want to override these values, create a text file (typically saved in the params folder with the extension .par) that contains comma-separated values paramname,value, like this:

- `length`, 5
- `trialsperblock`, 25
- `numblocks`, 5

Then, create a parameter object using `CreateParameters`:

```r
gParams <- CreateParameters(pairlist, filename)
```

Any values in the .par file will override the values in the default list. PEBL tries to convert text values to numbers, and the value will be a number whenever the round-trip from text-to-number-to-text does not change the original value. Thus, avoid using floating-point values for parameters, and you may need to write “0.1” instead of “.1” if you do.

The PEBL launcher offers a way to set parameters. To do so, it needs more information, including the default values and a description. It looks for a .schema file in the params folder with the same name as the experiment. This file uses the `|` character to separate field (this allows you to use commas in the description):

- `length|10|The number of words per trial`
- `trialsperblock|15|Trials in each block`
- `numblock|3|Number of blocks`
Chapter 4. Overview of Object Subsystems

Notice that quotes are not used in these files. The values in this file are ONLY relevant to the PEBL launcher. These are used to reset the values in a .par file or tell the experiment what typical values are. They can differ from the default values in the .pbl file, but for clarity they should not.

4.23 Errors and Warnings

PEBL does a great deal of error-checking to ensure that your program will run. If you crash with a segmentation fault, this is an error and you should report it. When a fatal error or non-fatal warning occurs, PEBL attempts to identify the location in your input file that led to the warning. On Linux, the warning and this location are printed to the command-line upon exit; on MS Windows or if you are using the launcher on any platform, they are printed to the file stderr.txt.

You can do error checking in your own scripts with the SignalFatalError() function. This is especially useful in combination with the functions testing the type of object passed into the function. To ensure proper processing and ease of debugging, test the format of an argument passed into a function:

define MyFunction(par)
{
    if(not IsList(par))
    {
        SignalFatalError("MyFunction passed a non-list variable.")
    }
    ##Do other stuff here
}

4.24 Paths and Path Searching

Numerous functions and objects open files on your computer to read in information such as graphics, sounds, fonts, program files, and text files. When you attempt to open a file, PEBL will search in a number of places, in this order:

- The (current) working directory
- The directory of each file specified in the command line arguments
- media/fonts
- media/sounds
- media/images
- media/text

You can also specify other paths to be searched by specifying them on the command line. Be sure to end the directory with whatever is appropriate for your platform, e.g. "\" on Microsoft Windows or "/" on Linux.
Chapter 4. Overview of Object Subsystems

On OSX, as of version 0.12, the media/ directory is located within the Resources/ subdirectory of the application package.

### 4.25 Provided Media Files

PEBL comes with various media files that can be specified from any script without including the complete path. If a user’s file has the same name, it will be loaded before the PEBL-provided version. Table 4.2 describes the files included.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In <code>media/fonts/</code>:</td>
<td></td>
</tr>
<tr>
<td>Listing of fonts appears in Table 4.1</td>
<td></td>
</tr>
<tr>
<td>In <code>media/images/</code>:</td>
<td></td>
</tr>
<tr>
<td>pebl.bmp</td>
<td>Demonstration bitmap image</td>
</tr>
<tr>
<td>pebl.png</td>
<td>Demonstration PNG image</td>
</tr>
<tr>
<td>smiley-small.png</td>
<td>25x25 smiley face</td>
</tr>
<tr>
<td>frowney-small.png</td>
<td>25x25 frowney face</td>
</tr>
<tr>
<td>smiley-large.png</td>
<td>100x100 smiley face</td>
</tr>
<tr>
<td>frowney-large.png</td>
<td>100x100 frowney face</td>
</tr>
<tr>
<td>plus.png</td>
<td>A green plus sign</td>
</tr>
<tr>
<td>x.png</td>
<td>A red x sign, matching the red plus</td>
</tr>
<tr>
<td>In <code>media/sounds/</code>:</td>
<td></td>
</tr>
<tr>
<td>buzz500ms.wav</td>
<td>A 500-ms buzzer</td>
</tr>
<tr>
<td>chirp1.wav</td>
<td>A chirp stimulus</td>
</tr>
<tr>
<td>boo.wav</td>
<td>A really bad booing sound</td>
</tr>
<tr>
<td>cheer.wav</td>
<td>A pretty lame cheering sound</td>
</tr>
<tr>
<td>beep.wav</td>
<td>A simple beep</td>
</tr>
<tr>
<td>boo.wav</td>
<td>Boo—useful for errors</td>
</tr>
<tr>
<td>cheer.wav</td>
<td>A cheer—useful for correct feedback</td>
</tr>
<tr>
<td>kaching.wav</td>
<td>Sound of a coin in a jar</td>
</tr>
<tr>
<td>knock.wav</td>
<td>simple knocking/click sound</td>
</tr>
<tr>
<td>0.wav through 9.wav</td>
<td>Recording of numerals, used in digit span and others</td>
</tr>
<tr>
<td>correct.wav</td>
<td>correct feedback</td>
</tr>
<tr>
<td>incorrect.wav</td>
<td>Incorrect feedback</td>
</tr>
<tr>
<td>H,R,N,K,X,Y,W.wav</td>
<td>Female voice letters for n-back</td>
</tr>
<tr>
<td>In <code>media/text/</code>:</td>
<td></td>
</tr>
</tbody>
</table>

40
Chapter 4. Overview of Object Subsystems

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consonants.txt</td>
<td>List of all consonants, both cases</td>
</tr>
<tr>
<td>Digits.txt</td>
<td>List of digits 0-9</td>
</tr>
<tr>
<td>DigitNames.txt</td>
<td>List of digit names</td>
</tr>
<tr>
<td>Letters.txt</td>
<td>All letters, both cases</td>
</tr>
<tr>
<td>Lowercase.txt</td>
<td>Lowercase letters</td>
</tr>
<tr>
<td>LowercaseConsonants.txt</td>
<td>Lowercase Consonants</td>
</tr>
<tr>
<td>LowercaseVowels.txt</td>
<td>Lowercase Vowels</td>
</tr>
<tr>
<td>Uppercase.txt</td>
<td>Uppercase Letters</td>
</tr>
<tr>
<td>UppercaseConsonants.txt</td>
<td>Uppercase Consonants</td>
</tr>
<tr>
<td>UppercaseVowels.txt</td>
<td>Uppercase Vowels</td>
</tr>
<tr>
<td>Vowels.txt</td>
<td>Vowels (both cases)</td>
</tr>
</tbody>
</table>

Additionally, the PEBL Project distributes a number of other media files separately from the base system. These are available for separate download on the pebl website (http://pebl.sourceforge.net), and include a set of images (including shapes and sorting-task cards), and a set of auditory recordings (including beeps, the digits 0-10, and a few other things).

4.26 Special Variables

There are a number of special variables that be set by PEBL, and can later be accessed by an experiment. These are described in table 4.3.
Chapter 4. Overview of Object Subsystems

Table 4.3: Special Variables in PEBL

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>gKeepLooping</td>
<td>Controls continued execution in event loop (unused).</td>
</tr>
<tr>
<td>gSleepEasy</td>
<td>Sets 'busy-waiting' to be either on or off. Busy-waiting can improve timing, but is often not needed and pegs CPU.</td>
</tr>
<tr>
<td>gVideoWidth</td>
<td>The width in pixels of the display (set by default or command-line option). Changing this before calling MakeWindow will change display width, if that width is available.</td>
</tr>
<tr>
<td>gVideoHeight</td>
<td>The height in pixels of the display (set by default or command-line). Change this before using MakeWindow() to change the display height</td>
</tr>
<tr>
<td>gVideoDepth</td>
<td>The bit depth of the video.</td>
</tr>
<tr>
<td>gSubNum</td>
<td>A global variable set to whatever follows the --s or --S command-line argument. Defaults to &quot;0&quot;.</td>
</tr>
<tr>
<td>gLanguage</td>
<td>A global variable specified on the command line which can be used by a script to target a specific language. Defaults to 'en'.</td>
</tr>
<tr>
<td>gQuote</td>
<td>A quotation mark: &quot;. Use it to add quotes in text.</td>
</tr>
<tr>
<td>gClick</td>
<td>[x,y] location last click in WaitForClickOnTarget.</td>
</tr>
<tr>
<td>gPEBLBaseFont</td>
<td>Name of the default font to use in helper functions and most battery tasks. By default, set to “DejaVuSans.ttf”. Change to override.</td>
</tr>
<tr>
<td>gPEBLBaseFontMono</td>
<td>Name of the default mono-spaced font</td>
</tr>
<tr>
<td>gPEBLBaseFontSerif</td>
<td>Name of the default serif font</td>
</tr>
<tr>
<td></td>
<td>By default, it is set to “DejaVuSerif.ttf”.</td>
</tr>
</tbody>
</table>
Chapter 5

Function Quick Reference

Table 5.1 lists the functions available for use with PEBL. Those that are unimplemented are noted as such. If you want the functionality of an unimplemented function, or want functionality not provided in any of these functions, contact us, or better yet, contribute to the PEBL project by implementing the function yourself.

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log10</td>
<td>&lt;num&gt;</td>
<td>Log base 10 of &lt;num&gt;</td>
</tr>
<tr>
<td>Log2</td>
<td>&lt;num&gt;</td>
<td>Log base 2 of &lt;num&gt;</td>
</tr>
<tr>
<td>Ln</td>
<td>&lt;num&gt;</td>
<td>Natural log of &lt;num&gt;</td>
</tr>
<tr>
<td>LogN</td>
<td>&lt;num&gt; &lt;base&gt;</td>
<td>Log base &lt;base&gt; of &lt;num&gt;</td>
</tr>
<tr>
<td>Exp</td>
<td>&lt;pow&gt;</td>
<td>e to the power of &lt;pow&gt;</td>
</tr>
<tr>
<td>Pow</td>
<td>&lt;num&gt; &lt;pow&gt;</td>
<td>&lt;num&gt; to the power of &lt;pow&gt;</td>
</tr>
<tr>
<td>Sqrt</td>
<td>&lt;num&gt;</td>
<td>Square root of &lt;num&gt;</td>
</tr>
<tr>
<td>NthRoot</td>
<td>&lt;num&gt; &lt;root&gt;</td>
<td>&lt;num&gt; to the power of 1/root</td>
</tr>
<tr>
<td>Tan</td>
<td>&lt;deg&gt;</td>
<td>Tangent of &lt;deg&gt; degrees</td>
</tr>
<tr>
<td>Sin</td>
<td>&lt;deg&gt;</td>
<td>Sine of &lt;deg&gt; degrees</td>
</tr>
<tr>
<td>Cos</td>
<td>&lt;deg&gt;</td>
<td>Cosine of &lt;deg&gt; degrees</td>
</tr>
<tr>
<td>ATan</td>
<td>&lt;num&gt;</td>
<td>Inverse Tan of &lt;num&gt;, in degrees</td>
</tr>
<tr>
<td>ASin</td>
<td>&lt;num&gt;</td>
<td>Inverse Sine of &lt;num&gt;, in degrees</td>
</tr>
<tr>
<td>ACos</td>
<td>&lt;num&gt;</td>
<td>Inverse Cosine of &lt;num&gt;, in degrees</td>
</tr>
<tr>
<td>DegToRad</td>
<td>&lt;deg&gt;</td>
<td>Converts degrees to radians</td>
</tr>
<tr>
<td>RadToDeg</td>
<td>&lt;rad&gt;</td>
<td>Converts radians to degrees</td>
</tr>
<tr>
<td>Round</td>
<td>&lt;num&gt;, (optional) &lt;precision&gt;</td>
<td>Rounds &lt;num&gt; to nearest integer, or optionally power of 1/ten precision.</td>
</tr>
<tr>
<td>Floor</td>
<td>&lt;num&gt;</td>
<td>Rounds &lt;num&gt; down to the next integer</td>
</tr>
<tr>
<td>Ceiling</td>
<td>&lt;num&gt;</td>
<td>Rounds &lt;num&gt; up to the next integer</td>
</tr>
<tr>
<td>Name</td>
<td>Arguments</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AbsFloor</td>
<td>&lt;num&gt;</td>
<td>Rounds &lt;num&gt; toward 0 to an integer</td>
</tr>
<tr>
<td>Mod</td>
<td>&lt;num&gt; &lt;mod&gt;</td>
<td>Returns &lt;num&gt; mod &lt;mod&gt; or remainder of &lt;num&gt;/&lt;mod&gt;</td>
</tr>
<tr>
<td>Div</td>
<td>&lt;num&gt; &lt;mod&gt;</td>
<td>Returns round(&lt;num&gt;/&lt;mod&gt;)</td>
</tr>
<tr>
<td>ToInteger</td>
<td>&lt;num&gt;</td>
<td>Rounds a number to an integer, and changes internal representation</td>
</tr>
<tr>
<td>ToFloat</td>
<td>&lt;num&gt;</td>
<td>Converts number to internal floating-point representation</td>
</tr>
<tr>
<td>ToNumber</td>
<td>&lt;&gt;</td>
<td>Converst to a number</td>
</tr>
<tr>
<td>ToString</td>
<td>&lt;num&gt;</td>
<td>Converts a numerical value to a string representation</td>
</tr>
<tr>
<td>Sign</td>
<td>&lt;num&gt;</td>
<td>Returns +1 or -1, depending on sign of argument</td>
</tr>
<tr>
<td>Abs</td>
<td>&lt;num&gt;</td>
<td>Returns the absolute value of the number</td>
</tr>
<tr>
<td>CumNormInv</td>
<td>&lt;p&gt;</td>
<td>Returns accurate numerical approximation of cumulative normal inverse.</td>
</tr>
<tr>
<td>NormalDensity</td>
<td>&lt;x&gt;</td>
<td>Returns density of standard normal distribution.</td>
</tr>
<tr>
<td>SDTDPriime</td>
<td>&lt;hr&gt;,&lt;far&gt;</td>
<td>Computes SDT dprime.</td>
</tr>
<tr>
<td>SDTBeta</td>
<td>&lt;hr&gt;,&lt;far&gt;</td>
<td>Computes SDT beta.</td>
</tr>
<tr>
<td>Order</td>
<td>&lt;list&gt;</td>
<td>Returns a list of integers representing the order of &lt;list&gt;</td>
</tr>
<tr>
<td>Rank</td>
<td>&lt;list&gt;</td>
<td>Returns integers representing the ranked indices of the numbers of &lt;list&gt;</td>
</tr>
<tr>
<td>Median</td>
<td>&lt;list&gt;</td>
<td>Returns the median value of the numbers in &lt;list&gt;</td>
</tr>
<tr>
<td>Min</td>
<td>&lt;list&gt;</td>
<td>Returns the smallest of &lt;list&gt;</td>
</tr>
<tr>
<td>Max</td>
<td>&lt;list&gt;</td>
<td>Returns the largest of &lt;list&gt;</td>
</tr>
<tr>
<td>Bound</td>
<td>&lt;val&gt;, &lt;min&gt;,&lt;max&gt;</td>
<td>Returns val, bounded by min and max.</td>
</tr>
<tr>
<td>StDev</td>
<td>&lt;list&gt;</td>
<td>Returns the standard dev of &lt;list&gt;</td>
</tr>
<tr>
<td>Sum</td>
<td>&lt;list&gt;</td>
<td>Returns the sum of the numbers in &lt;list&gt;</td>
</tr>
<tr>
<td>Median</td>
<td>&lt;list&gt;</td>
<td>Returns the median of a set of values</td>
</tr>
<tr>
<td>Quantile</td>
<td>&lt;list&gt; &lt;num&gt;</td>
<td>Returns the &lt;num&gt; quantile of the numbers in &lt;list&gt;</td>
</tr>
<tr>
<td>SummaryStats</td>
<td>&lt;data&gt;,&lt;cond&gt;</td>
<td>Returns statistics (cond, N, median, mean, sd) computed on data for each</td>
</tr>
<tr>
<td></td>
<td></td>
<td>distinct value of &lt;cond&gt;</td>
</tr>
<tr>
<td>SeedRNG</td>
<td>&lt;num&gt;</td>
<td>Seeds the random number generator with &lt;num&gt; to reproduce a random sequence</td>
</tr>
</tbody>
</table>

44
<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomizeTimer</td>
<td>-</td>
<td>Seeds the RNG with the current time</td>
</tr>
<tr>
<td>Random</td>
<td>-</td>
<td>Returns a random number between 0 and 1</td>
</tr>
<tr>
<td>RandomDiscrete</td>
<td>&lt;num&gt;</td>
<td>Returns a random integer between 1 and &lt;num&gt;</td>
</tr>
<tr>
<td>RandomUniform</td>
<td>&lt;num&gt;</td>
<td>Returns a random floating-point number between 0 and &lt;num&gt;</td>
</tr>
<tr>
<td>RandomNormal</td>
<td>&lt;mean&gt; &lt;stdev&gt;</td>
<td>Returns a random number according to the standard normal distribution with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;mean&gt; and &lt;stdev&gt;</td>
</tr>
<tr>
<td>RandomExponential</td>
<td>&lt;mean&gt;</td>
<td>Returns a random number according to exponential distribution with mean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;mean&gt; (or decay 1/&lt;mean&gt;)</td>
</tr>
<tr>
<td>RandomLogistic</td>
<td>&lt;p&gt;</td>
<td>Returns a random number according to the logistic distribution with parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;p&gt;</td>
</tr>
<tr>
<td>RandomLogNormal</td>
<td>&lt;median&gt; &lt;spread&gt;</td>
<td>Returns a random number according to the log-normal distribution with pa-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rameters &lt;median&gt; and &lt;spread&gt;</td>
</tr>
<tr>
<td>RandomBinomial</td>
<td>&lt;p&gt; &lt;n&gt;</td>
<td>Returns a random number according to the Binomial distribution with proba-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bility &lt;p&gt; and repetitions &lt;n&gt;</td>
</tr>
<tr>
<td>RandomBernoulli</td>
<td>&lt;p&gt;</td>
<td>Returns 0 with probability (1-&lt;p&gt;) and 1 with probability &lt;p&gt;</td>
</tr>
<tr>
<td>ZoomPoints</td>
<td>&lt;[xs,yy]&gt;</td>
<td>Zooms a set of points in 2 directions</td>
</tr>
<tr>
<td></td>
<td>&lt;xzoom&gt; &lt;yzoom&gt;</td>
<td></td>
</tr>
<tr>
<td>ReflectPoints</td>
<td>&lt;[xs,yy]&gt;</td>
<td>Reflects points on vertical axis</td>
</tr>
<tr>
<td>RotatePoints</td>
<td>&lt;[xs,yy]&gt; &lt;angle&gt;</td>
<td>Rotates point &lt;angle&gt; degrees</td>
</tr>
<tr>
<td>GetAngle</td>
<td>&lt;x&gt;,&lt;y&gt;</td>
<td>Returns the angle in degrees of a vector.</td>
</tr>
<tr>
<td>Dist</td>
<td>&lt;x1,y1&gt;,&lt;x2,y2&gt;</td>
<td>Returns distance between two points.</td>
</tr>
<tr>
<td>ToRight</td>
<td>&lt;p1,p2,p3&gt;</td>
<td>Determines whether p3 is to the right of line p1p2</td>
</tr>
<tr>
<td>GetAngle3</td>
<td>&lt;a,b,c&gt;</td>
<td>Gets angle abc.</td>
</tr>
<tr>
<td>SegmentsIntersect</td>
<td>&lt;ax,ay,bx,by,cx,cy,dx,dy&gt;</td>
<td>Determines whether line segment ax intersects cd.</td>
</tr>
<tr>
<td>NonOverlapLayout</td>
<td>&lt;xmin,xmax,ymin,ymax,tol,num&gt;</td>
<td>Creates a set of num points that don’t overlap, but fails gracefully</td>
</tr>
<tr>
<td>LayoutGrid</td>
<td>&lt;minx,maxx,miny,maxy,height,width,vertical&gt;</td>
<td>Creates [xy] pairs in a grid for graphical layout</td>
</tr>
</tbody>
</table>
## This is an image of a page from a document

### Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File/Network/Device Functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Print</td>
<td>&lt;value&gt;</td>
<td>Prints &lt;value&gt; to stdout, appending a new line afterwards. <code>stdout</code> is the console (in Linux) or the file <code>stdout.txt</code> (in Windows)</td>
</tr>
<tr>
<td>Print_</td>
<td>&lt;value&gt;</td>
<td>Prints &lt;value&gt; to stdout, without appending a newline afterwards</td>
</tr>
<tr>
<td>PrintList</td>
<td>&lt;value&gt;</td>
<td>Prints &lt;list&gt;, getting rid of ']', ']' and ',' characters.</td>
</tr>
<tr>
<td>Format</td>
<td>&lt;object&gt; &lt;size&gt;</td>
<td>Prints a number with specified spaces by truncating or padding</td>
</tr>
<tr>
<td>ZeroPad</td>
<td>&lt;number&gt; &lt;size&gt;</td>
<td>Pads the beginning of a number with 0s so the number is size long</td>
</tr>
<tr>
<td>FileOpenRead</td>
<td>&lt;filename&gt;</td>
<td>Opens a filename, returning a stream to be used for reading information</td>
</tr>
<tr>
<td>FileOpenWrite</td>
<td>&lt;filename&gt;</td>
<td>Opens a filename, returning a stream that can be used for writing information. Creates new file if file already exists</td>
</tr>
<tr>
<td>FileOpenOverwrite</td>
<td>&lt;filename&gt;</td>
<td>Opens a filename, returning a stream that can be used for writing information. Overwrites if file already exists</td>
</tr>
<tr>
<td>FileOpenAppend</td>
<td>&lt;filename&gt;</td>
<td>Opens a filename, returning a stream that can be used for writing information. Appends if the file already exists, opens if file does not</td>
</tr>
<tr>
<td>FileClose</td>
<td>&lt;filestream&gt;</td>
<td>Closes a filestream variable. Pass the variable name, not the filename</td>
</tr>
<tr>
<td>FilePrint</td>
<td>&lt;filestream&gt; &lt;value&gt;</td>
<td>Like <code>Print</code>, but to a file.</td>
</tr>
<tr>
<td>FilePrint_</td>
<td>&lt;filestream&gt; &lt;value&gt;</td>
<td>Like <code>Print_</code>, but to a file.</td>
</tr>
<tr>
<td>FilePrintList</td>
<td>&lt;file&gt; &lt;list&gt;</td>
<td>Prints &lt;list&gt; to &lt;file&gt;, getting rid of ']', ']' and ',' characters.</td>
</tr>
<tr>
<td>FileReadCharacter</td>
<td>&lt;filestream&gt;</td>
<td>Reads and returns a single character from a filestream</td>
</tr>
<tr>
<td>FileReadWord</td>
<td>&lt;filestream&gt;</td>
<td>Reads and returns a ‘word’ from a file; the next connected stream of characters not including a ‘ ’ or a newline. Will not read newline characters</td>
</tr>
</tbody>
</table>
## Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FileReadLine</code></td>
<td><code>&lt;filestream&gt;</code></td>
<td>Reads and returns a line from a file; all characters up until the next newline or the end of the file</td>
</tr>
<tr>
<td><code>FileReadList</code></td>
<td><code>&lt;filename&gt;</code></td>
<td>Given a filename, will open it, read in all the items into a list (one item per line), and close the file afterwards</td>
</tr>
<tr>
<td><code>FileReadTable</code></td>
<td><code>&lt;filename&gt;</code></td>
<td>Like <code>FileReadList</code>, but reads in tables. Optionally, specify a token separator</td>
</tr>
<tr>
<td><code>GetNewDatafile</code></td>
<td><code>&lt;id-code&gt;, &lt;window&gt;, &lt;basename&gt;, &lt;extension&gt;, &lt;header&gt;</code></td>
<td>Opens a data file in subnum directory</td>
</tr>
<tr>
<td><code>ReadCSV</code></td>
<td><code>&lt;filename&gt;</code></td>
<td>Opens a csv file returning a table with its elements</td>
</tr>
<tr>
<td><code>FileReadText</code></td>
<td><code>&lt;filename&gt;</code></td>
<td>Reads all of the text in the file into a variable</td>
</tr>
<tr>
<td><code>EndOfLine</code></td>
<td><code>&lt;filestream&gt;</code></td>
<td>Returns true if at end of line</td>
</tr>
<tr>
<td><code>EndOfFile</code></td>
<td><code>&lt;filestream&gt;</code></td>
<td>Returns true if at the end of a file</td>
</tr>
<tr>
<td><code>GetDirectoryListing</code></td>
<td><code>&lt;path&gt;</code></td>
<td>Returns a list of all the files/subdirectories in a path</td>
</tr>
<tr>
<td><code>FileExists</code></td>
<td><code>&lt;path&gt;</code></td>
<td>Checks whether a file exists</td>
</tr>
<tr>
<td><code>IsDirectory</code></td>
<td><code>&lt;path&gt;</code></td>
<td>Checks whether a file is a directory</td>
</tr>
<tr>
<td><code>MakeDirectory</code></td>
<td><code>&lt;path&gt;, &lt;dirname&gt;</code></td>
<td>Creates a directory in path</td>
</tr>
<tr>
<td><code>RemoveFile</code></td>
<td><code>&lt;file&gt;</code></td>
<td>Removes and deletes a file</td>
</tr>
<tr>
<td><code>AppendFile</code></td>
<td><code>&lt;file1&gt;, &lt;file2&gt;</code></td>
<td>Appends a file2 to file1</td>
</tr>
<tr>
<td><code>DeleteFile</code></td>
<td><code>&lt;file1&gt;</code></td>
<td>Deletes a file</td>
</tr>
<tr>
<td><code>ConnectToIP</code></td>
<td><code>&lt;ip&gt; &lt;port&gt;</code></td>
<td>Connects to a port on another computer, returning network object.</td>
</tr>
<tr>
<td><code>ConnectToHost</code></td>
<td><code>&lt;hostname&gt; &lt;port&gt;</code></td>
<td>Connects to a port on another computer, returning network object.</td>
</tr>
<tr>
<td><code>WaitForNetworkConnection</code></td>
<td><code>&lt;port&gt;</code></td>
<td>Listens on a port until another computer connects, returning a network object</td>
</tr>
<tr>
<td><code>CloseNetworkConnection</code></td>
<td><code>&lt;network&gt;</code></td>
<td>Closes network connection</td>
</tr>
<tr>
<td><code>SendData</code></td>
<td><code>&lt;network&gt; &lt;datastring&gt;</code></td>
<td>Sends a data string over connection</td>
</tr>
<tr>
<td><code>GetData</code></td>
<td><code>&lt;network&gt; &lt;length&gt;</code></td>
<td>Return a string from network connection</td>
</tr>
<tr>
<td><code>ConvertIPString</code></td>
<td><code>&lt;ip-as-string&gt;</code></td>
<td>Converts an ip-number-as-string to usable address</td>
</tr>
</tbody>
</table>
## Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenNetworkListener</td>
<td>&lt;port&gt;</td>
<td>Opens a port for listening</td>
</tr>
<tr>
<td>CheckForNetworkConnection</td>
<td>&lt;network&gt;</td>
<td>Checks for incoming connection</td>
</tr>
<tr>
<td>GetHTTPFile</td>
<td>&lt;server&gt;, &lt;file&gt;, &lt;outputfile&gt;</td>
<td>Gets and saves a file from a website</td>
</tr>
<tr>
<td>GetHTTPText</td>
<td>&lt;server&gt;, &lt;file&gt;</td>
<td>Gets a file from a website and saves it to a variable.</td>
</tr>
<tr>
<td>PostHTTP</td>
<td>&lt;host&gt;, &lt;page&gt;, &lt;file&gt;, &lt;headers&gt;, &lt;content&gt;</td>
<td>Post to a server form.</td>
</tr>
<tr>
<td>MD5Sum</td>
<td>&lt;text&gt;</td>
<td>Computes MD5 checksum on text</td>
</tr>
<tr>
<td>MD5File</td>
<td>&lt;filename&gt;</td>
<td>Computes MD5_checksum on file.</td>
</tr>
<tr>
<td>WritePNG</td>
<td>&lt;filename&gt;, &lt;object&gt;</td>
<td>Makes a .png from a window or object.</td>
</tr>
<tr>
<td>GetNumJoysticks</td>
<td>no argument</td>
<td>Determines how many joysticks are available</td>
</tr>
<tr>
<td>OpenJoystick</td>
<td>joystick_id</td>
<td>Gets a joystick object</td>
</tr>
<tr>
<td>GetNumJoystickAxes</td>
<td>joystick_object</td>
<td>Counts how many axes on a joystick</td>
</tr>
<tr>
<td>GetNumJoystickBalls</td>
<td>joystick_object</td>
<td>Counts how many balls on a joystick</td>
</tr>
<tr>
<td>GetNumJoystickButtons</td>
<td>joystick_object</td>
<td>Counts how many buttons on a joystick</td>
</tr>
<tr>
<td>GetNumJoystickHats</td>
<td>joystick_object</td>
<td>Counts how many hats on a joystick</td>
</tr>
<tr>
<td>GetJoystickAxisState</td>
<td>joystick_object, axis_id</td>
<td>Gets the state of a joystick axis</td>
</tr>
<tr>
<td>GetJoystickHatState</td>
<td>joystick_object, hat_id</td>
<td>Gets the state of a joystick hat</td>
</tr>
<tr>
<td>GetJoystickButtonState</td>
<td>joystick_object, button_id</td>
<td>Gets the state of a joystick button</td>
</tr>
<tr>
<td>GetJoystickBallState</td>
<td>joystick_object, ball_id</td>
<td>Gets the state of a joystick ball</td>
</tr>
<tr>
<td>OpenCOMPort</td>
<td>&lt;portnum&gt;, &lt;baud&gt;</td>
<td>Opens a serial (com) port</td>
</tr>
<tr>
<td>COMPortGetByte</td>
<td>&lt;port&gt;</td>
<td>Gets a byte from the comport</td>
</tr>
<tr>
<td>COMPortSendByte</td>
<td>&lt;port&gt;, &lt;byte&gt;</td>
<td>Sends a character to the comport</td>
</tr>
<tr>
<td>OpenPPort</td>
<td>&lt;portname&gt;</td>
<td>Opens parallel port</td>
</tr>
<tr>
<td>SetPPortMode</td>
<td>&lt;port&gt;, &lt;mode&gt;</td>
<td>Sets parallel port mode (input/output)</td>
</tr>
<tr>
<td>SetPPortState</td>
<td>&lt;port&gt;, &lt;state&gt;</td>
<td>Sets parallel port state</td>
</tr>
</tbody>
</table>
## Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPPortState</td>
<td>&lt;port&gt;</td>
<td>Gets state of parallel port data bits</td>
</tr>
</tbody>
</table>

### Graphical Objects Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakeWindow</td>
<td>&lt;colorname&gt;</td>
<td>Creates main window, in color named by argument, or grey if no argument is named</td>
</tr>
<tr>
<td>MakeImage</td>
<td>&lt;filename&gt;</td>
<td>Creates an image by reading in an image file (jpg, gif, png, bmp, etc.)</td>
</tr>
<tr>
<td>MakeLabel</td>
<td>&lt;text&gt; &lt;font&gt;</td>
<td>Creates a single line of text filled with &lt;text&gt; written in font &lt;font&gt;</td>
</tr>
<tr>
<td>MakeTextBox</td>
<td>&lt;text&gt; &lt;font&gt; &lt;width&gt; &lt;height&gt;</td>
<td>Creates a sized box filled with &lt;text&gt; written in font &lt;font&gt;</td>
</tr>
<tr>
<td>MakeCanvas</td>
<td>&lt;width&gt;, &lt;height&gt;, &lt;color&gt;</td>
<td>Creates a blank canvas to add objects to or draw on</td>
</tr>
<tr>
<td>ResetCanvas</td>
<td>&lt;canvas&gt;</td>
<td>Resets a canvas to its background, erasing anything drawn on the canvas</td>
</tr>
<tr>
<td>EasyLabel</td>
<td>&lt;text&gt; &lt;x&gt; &lt;y&gt; &lt;win&gt; &lt;fontsize&gt;</td>
<td>Creates a single line of text and adds it to win at &lt;x&gt; &lt;y&gt;</td>
</tr>
<tr>
<td>EasyTextBox</td>
<td>&lt;text&gt; &lt;x&gt; &lt;y&gt; &lt;win&gt; &lt;fontsize&gt; &lt;width&gt; &lt;height&gt;</td>
<td>Creates a textbox and adds it to &lt;win&gt; at &lt;x&gt; &lt;y&gt;</td>
</tr>
<tr>
<td>MakeColor</td>
<td>&lt;colorname&gt;</td>
<td>Creates a color based on a color name</td>
</tr>
<tr>
<td>MakeColorRGB</td>
<td>&lt;red&gt; &lt;green&gt; &lt;blue&gt;</td>
<td>Creates a color based on red, green, and blue values</td>
</tr>
<tr>
<td>RGBToHSV</td>
<td>&lt;color&gt;</td>
<td>Converts a color to HSV triple</td>
</tr>
<tr>
<td>MakeFont</td>
<td>&lt;ttf_filename&gt; &lt;style&gt; &lt;size&gt; &lt;fgcolor&gt; &lt;bgcolor&gt; &lt;anti-aliased&gt;</td>
<td>Creates a font which can be used to make labels</td>
</tr>
<tr>
<td>SetCursorPosition</td>
<td>&lt;textbox&gt; &lt;position&gt;</td>
<td>Move the editing cursor in a textbox</td>
</tr>
<tr>
<td>GetCursorPosition</td>
<td>&lt;textbox&gt; &lt;position&gt;</td>
<td>Gets the position of the editing cursor</td>
</tr>
<tr>
<td>SetEditable</td>
<td>&lt;textbox&gt; &lt;status&gt;</td>
<td>Turns on or off the editing cursor</td>
</tr>
<tr>
<td>GetTextBoxCursorFromClick</td>
<td>&lt;relx&gt;,&lt;rely&gt;</td>
<td>Gets a cursor position (in characters) from a mouse click.</td>
</tr>
<tr>
<td>GetText</td>
<td>&lt;textobject&gt;</td>
<td>Returns the text in a textbox or label</td>
</tr>
</tbody>
</table>
## Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetInput</td>
<td>&lt;textbox&gt;, &lt;escape-key&gt;</td>
<td>Allows a textbox to be edited by user, returning its text when &lt;escape-key&gt; is pressed.</td>
</tr>
<tr>
<td>SetText</td>
<td>&lt;textobject&gt;, &lt;text&gt;</td>
<td>Sets the text in a textbox or label</td>
</tr>
<tr>
<td>SetFont</td>
<td>&lt;textobject&gt;, &lt;font&gt;</td>
<td>Changes the font of a text object</td>
</tr>
<tr>
<td>Move</td>
<td>&lt;object&gt; &lt;x&gt; &lt;y&gt;</td>
<td>Move an object (e.g., an image or a label to an x,y location)</td>
</tr>
<tr>
<td>MoveCorner</td>
<td>&lt;object&gt; &lt;x&gt; &lt;y&gt;</td>
<td>Moves an image or label by its upper corner.</td>
</tr>
<tr>
<td>GetSize</td>
<td>&lt;object&gt;</td>
<td>Returns a list of dimensions &lt;x,y&gt; of a graphical object.</td>
</tr>
<tr>
<td>AddObject</td>
<td>&lt;object&gt;, &lt;parent&gt;</td>
<td>Adds an object to a parent object (window)</td>
</tr>
<tr>
<td>RemoveObject</td>
<td>&lt;object&gt;, &lt;parent&gt;</td>
<td>Removes an object from a parent window</td>
</tr>
<tr>
<td>Show</td>
<td>&lt;object&gt;, &lt;parent&gt;</td>
<td>Shows an object</td>
</tr>
<tr>
<td>Hide</td>
<td>&lt;object&gt;</td>
<td>Hides an object</td>
</tr>
<tr>
<td>ShowCursor</td>
<td>&lt;object&gt;</td>
<td>Hides or show mouse cursor.</td>
</tr>
<tr>
<td>GetMouseCursorPosition</td>
<td></td>
<td>Gets [x,y] position of mouse</td>
</tr>
<tr>
<td>GetMouseState</td>
<td></td>
<td>Gets [x,y,b1,b2,b3] list of mouse state, including button states.</td>
</tr>
<tr>
<td>SetMouseCursorPosition</td>
<td></td>
<td>Sets x,y position of mouse</td>
</tr>
<tr>
<td>Draw</td>
<td>&lt;object&gt; or no argument</td>
<td>Redraws a widget and its children</td>
</tr>
<tr>
<td>DrawFor</td>
<td>&lt;object&gt;, &lt;cycles&gt;</td>
<td>Draws for exactly &lt;cycles&gt; cycles, then returns</td>
</tr>
<tr>
<td>Circle</td>
<td>&lt;x&gt; &lt;y&gt; &lt;r&gt; &lt;color&gt; &lt;filled&gt;</td>
<td>Creates circle with radius r centered at position x,y</td>
</tr>
<tr>
<td>Ellipse</td>
<td>&lt;x&gt; &lt;y&gt; &lt;rx&gt; &lt;ry&gt; &lt;color&gt; &lt;filled&gt;</td>
<td>Creates ellipse with radii rx and ry centered at position x,y</td>
</tr>
<tr>
<td>Square</td>
<td>&lt;x&gt; &lt;y&gt; &lt;size&gt; &lt;color&gt; &lt;filled&gt;</td>
<td>Creates square with width size centered at position x,y</td>
</tr>
<tr>
<td>Rectangle</td>
<td>&lt;x&gt; &lt;y&gt; &lt;dx&gt; &lt;dy&gt; &lt;color&gt; &lt;filled&gt;</td>
<td>Creates rectangle with size (dx, dy) centered at position x,y</td>
</tr>
<tr>
<td>Line</td>
<td>&lt;x&gt; &lt;y&gt; &lt;dx&gt; &lt;dy&gt; &lt;color&gt;</td>
<td>Creates line starting at x,y and ending at x+dx, y+dy</td>
</tr>
</tbody>
</table>
## Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polygon</td>
<td><code>&lt;x&gt; &lt;y&gt; &lt;xpoints&gt; &lt;ypoints&gt; &lt;color&gt; &lt;filled&gt;</code></td>
<td>Creates polygon centered at x,y with relative points <code>xpoints</code>, <code>ypoints</code></td>
</tr>
<tr>
<td>Bezier</td>
<td><code>&lt;x&gt; &lt;y&gt; &lt;xpoints&gt; &lt;ypoints&gt; &lt;steps&gt; &lt;color&gt;</code></td>
<td>Creates bezier curve centered at x,y with relative points</td>
</tr>
<tr>
<td>BlockE</td>
<td><code>&lt;x&gt; &lt;y&gt; &lt;h&gt; &lt;w&gt; &lt;thickness&gt; &lt;orientation&gt; &lt;color&gt;</code></td>
<td>Creates a block E as a useable polygon which can be added to a window directly.</td>
</tr>
<tr>
<td>Plus</td>
<td><code>&lt;x&gt; &lt;y&gt; &lt;size&gt; &lt;w&gt; &lt;color&gt;</code></td>
<td>Creates a plus sign as a useable polygon which can be added to a window directly.</td>
</tr>
<tr>
<td>MakeStarPoints</td>
<td><code>&lt;r_outer&gt; &lt;r_inner&gt; &lt;npeaks&gt;</code></td>
<td>Creates points for a star, which can then be fed to Polygon</td>
</tr>
<tr>
<td>MakeNGonPoints</td>
<td><code>&lt;radius&gt; &lt;npeaks&gt;</code></td>
<td>Creates points for a polygon, which can then be fed to Polygon</td>
</tr>
<tr>
<td>ThickLine</td>
<td><code>&lt;x1&gt; &lt;y1&gt; &lt;x2&gt; &lt;y2&gt; &lt;thickness&gt; &lt;color&gt;</code></td>
<td>Creates a thick line between two points</td>
</tr>
<tr>
<td>MakeAttneave</td>
<td><code>&lt;radius&gt;, &lt;mumpoints&gt;, &lt;minangle&gt;, &lt;maxangle&gt;</code></td>
<td>Makes a complex “Attneave” polygon</td>
</tr>
<tr>
<td>ConvexHull</td>
<td><code>&lt;list-of-pts&gt;</code></td>
<td>Returns a convex subset of points for a set</td>
</tr>
<tr>
<td>KaneszaSquare</td>
<td><code>&lt;squaresize&gt;, &lt;circleradius&gt;, &lt;fg&gt;, &lt;bg&gt;</code></td>
<td>Creates a 'Kanesza Square' stimulus.</td>
</tr>
<tr>
<td>KaneszaPolygon</td>
<td><code>&lt;points&gt;, &lt;circTF&gt;, &lt;circleradius&gt;, &lt;fg&gt;, &lt;bg&gt;, &lt;show&gt;</code></td>
<td>Create generic Kanesza polygon.</td>
</tr>
<tr>
<td>Inside</td>
<td><code>&lt;[x,y]&gt; &lt;object&gt;</code></td>
<td>Determines whether a point is inside a graphical object</td>
</tr>
<tr>
<td>SetPixel</td>
<td><code>x,y,color</code></td>
<td>Sets the color of a pixel on an image or canvas to <code>color</code></td>
</tr>
<tr>
<td>SetPoint</td>
<td><code>x,y,color</code></td>
<td>Sets the color of a pixel on an image or canvas to <code>color</code></td>
</tr>
<tr>
<td>GetPixelColor</td>
<td><code>&lt;obj&gt;, x,y</code></td>
<td>Gets the color of a specified pixel on a widget</td>
</tr>
</tbody>
</table>
### Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MakeGabor</strong></td>
<td><code>&lt;size&gt;, &lt;freq&gt;, &lt;sd&gt;, &lt;angle&gt;, &lt;phase&gt;, &lt;bglev&gt;</code></td>
<td>Creates a 'gabor patch' with specified parameters</td>
</tr>
<tr>
<td><strong>LoadSound</strong></td>
<td><code>&lt;filename&gt;</code></td>
<td>Loads a soundfile from the filename, returning a variable that can be played</td>
</tr>
<tr>
<td><strong>PlayForeground</strong></td>
<td><code>&lt;sound&gt;</code></td>
<td>Plays the sound 'in the foreground', not returning until the sound is complete</td>
</tr>
<tr>
<td><strong>PlayBackground</strong></td>
<td><code>&lt;sound&gt;</code></td>
<td>Plays the sound 'in the background', returning immediately</td>
</tr>
<tr>
<td><strong>Stop</strong></td>
<td><code>&lt;sound&gt;</code></td>
<td>Stops a sound playing in the background from playing</td>
</tr>
<tr>
<td><strong>MakeSineWave</strong></td>
<td><code>freq, duration, amplitude</code></td>
<td>Creates a pure sine wave.</td>
</tr>
<tr>
<td><strong>MakeAudioInputBuffer</strong></td>
<td><code>&lt;time-in-ms&gt;</code></td>
<td>Creates a buffer to record audio input</td>
</tr>
<tr>
<td><strong>SaveAudioToFile</strong></td>
<td><code>&lt;filename&gt;, &lt;buffer&gt;</code></td>
<td>Saves buffer to a .wav file format</td>
</tr>
<tr>
<td><strong>GetVocalResponseTime</strong></td>
<td><code>&lt;buffer&gt;, &lt;threshold&gt;, &lt;duration&gt;</code></td>
<td>A simple voice key</td>
</tr>
<tr>
<td><strong>LoadMovie</strong></td>
<td><code>&lt;movie_filename&gt;, &lt;window&gt;, &lt;width&gt;, &lt;height&gt;</code></td>
<td>Load a movie file</td>
</tr>
<tr>
<td><strong>LoadAudioFile</strong></td>
<td><code>&lt;audio_filename&gt;</code></td>
<td>Load an audio file</td>
</tr>
<tr>
<td><strong>PlayMovie</strong></td>
<td><code>&lt;movie&gt;</code></td>
<td>Plays a movie until its end</td>
</tr>
<tr>
<td><strong>StartPlayback</strong></td>
<td><code>&lt;movie&gt;</code></td>
<td>Initiates playback in background, updated with Wait()</td>
</tr>
<tr>
<td><strong>PausePlayback</strong></td>
<td><code>&lt;movie&gt;</code></td>
<td>Pauses playback of movie</td>
</tr>
<tr>
<td><strong>Graphical User Interface Functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GetTime</strong></td>
<td>&lt;&gt;</td>
<td>Gets a number, in milliseconds, representing the time since the PEBL program began running.</td>
</tr>
<tr>
<td><strong>MakeButton</strong></td>
<td><code>&lt;label&gt;, &lt;x&gt;, &lt;y&gt;, &lt;win&gt;, &lt;width&gt;</code></td>
<td>Makes a button for clicking on.</td>
</tr>
<tr>
<td><strong>PushButton</strong></td>
<td><code>&lt;button&gt;, &lt;[x, y]&gt;</code></td>
<td>Pushes a button and releases.</td>
</tr>
<tr>
<td><strong>MakeCheckBox</strong></td>
<td><code>&lt;label&gt;, &lt;x&gt;, &lt;y&gt;, &lt;win&gt;, &lt;width&gt;</code></td>
<td>Makes a two-state checkbox on a button background.</td>
</tr>
<tr>
<td><strong>ClickCheckBox</strong></td>
<td><code>&lt;checkbox&gt;, &lt;[x, y]&gt;</code></td>
<td>Handles checkbox click and updates state.</td>
</tr>
<tr>
<td><strong>SetCheckBox</strong></td>
<td><code>&lt;checkbox&gt;, &lt;state&gt;</code></td>
<td>Sets checkbox state.</td>
</tr>
<tr>
<td><strong>MakeScrollingTextBox</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakeScrollBox</td>
<td>&lt;opts&gt;, &lt;header&gt;, &lt;x&gt;, &lt;y&gt;, &lt;win&gt;, &lt;fontsize&gt;, &lt;width&gt;, &lt;height&gt;, &lt;selected&gt;</td>
<td>Make a scrolling selection box.</td>
</tr>
<tr>
<td>SetScrollingText</td>
<td>&lt;obj&gt;&lt;text&gt;, &lt;linewrap&gt;</td>
<td>Changes text of a scrolling text box.</td>
</tr>
<tr>
<td>UpdateScrollBar</td>
<td>&lt;obj&gt;</td>
<td>Recalculates scrollbar layout.</td>
</tr>
<tr>
<td>DrawScrollBar</td>
<td>&lt;obj&gt;</td>
<td>Draws a scrollbar.</td>
</tr>
<tr>
<td>ClickOnScrollBar</td>
<td>&lt;obj&gt;, &lt;click&gt;</td>
<td>Handles click on scrollbar.</td>
</tr>
<tr>
<td>PopUpMessageBox</td>
<td>&lt;label&gt;&lt;win&gt;</td>
<td>Makes a small message box at the mouse location.</td>
</tr>
<tr>
<td>PopUpEntryBox</td>
<td>&lt;label&gt;&lt;win&gt;&lt;[x,y]&gt;</td>
<td>Makes a small entry box at [xy] location.</td>
</tr>
<tr>
<td>MakePullDown</td>
<td>&lt;optionlist&gt;, &lt;x&gt;, &lt;y&gt;, &lt;win&gt;, &lt;fontsize&gt;, &lt;width&gt;, &lt;selected&gt;, &lt;functions&gt;</td>
<td>Make a pulldown selection box for a list.</td>
</tr>
<tr>
<td>PullDown</td>
<td>&lt;obj&gt;&lt;[x,y]&gt;</td>
<td>Handle click on a pulldown.</td>
</tr>
<tr>
<td>UpdatePullDown</td>
<td>&lt;obj&gt;, &lt;newlist&gt;</td>
<td>Updates the list of a pulldown.</td>
</tr>
<tr>
<td>DrawPullDown</td>
<td>&lt;obj&gt;</td>
<td>Redraws a pulldown if state changes.</td>
</tr>
<tr>
<td>MakeTextlist</td>
<td>&lt;list&gt;, &lt;listoffset&gt;, &lt;prebuffer&gt;</td>
<td>Creates a text body from a list.</td>
</tr>
<tr>
<td>InsideTB</td>
<td>{[xy]}&lt;obj&gt;</td>
<td>Determine inside for a text box-style object (location is upper left)</td>
</tr>
<tr>
<td>MakeMenu</td>
<td>&lt;header&gt;, &lt;x&gt;, &lt;y&gt;, &lt;win&gt;, &lt;fontsize&gt;, &lt;width&gt;, &lt;subitems&gt;, &lt;functions&gt;</td>
<td>Creates menu with suboptions.</td>
</tr>
</tbody>
</table>
### Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakeMenuItem</td>
<td><code>&lt;text&gt;</code>, <code>&lt;x&gt;</code>, <code>&lt;y&gt;</code>, <code>&lt;win&gt;</code>, <code>&lt;fontsize&gt;</code>, <code>&lt;width&gt;</code>, <code>&lt;function&gt;</code></td>
<td>Creates menu sub-item.</td>
</tr>
<tr>
<td>ClickOnMenu</td>
<td><code>&lt;obj&gt;</code>, <code>&lt;[x,y]&gt;</code></td>
<td>Handles menu click, calling the .clickOn function of menu.</td>
</tr>
<tr>
<td>OpenSubMenus</td>
<td><code>&lt;obj&gt;</code>, <code>&lt;[x,y]&gt;</code></td>
<td>Opens the sub-menus of a menu.</td>
</tr>
</tbody>
</table>

### Custom and Built-in Object Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintProperties</td>
<td><code>&lt;object&gt;</code></td>
<td>Prints a list of all available properties of an object (for debugging)</td>
</tr>
<tr>
<td>GetPropertyList</td>
<td><code>&lt;object&gt;</code></td>
<td>Gets a list of all the property names of an object</td>
</tr>
<tr>
<td>PropertyExists</td>
<td><code>&lt;object&gt;</code>, <code>&lt;prop&gt;</code></td>
<td>Determines whether a particular property exists</td>
</tr>
<tr>
<td>SetProperty</td>
<td><code>&lt;object&gt;</code>, <code>&lt;prop&gt;</code>, <code>&lt;value&gt;</code></td>
<td>Sets property of an object</td>
</tr>
<tr>
<td>GetProperty</td>
<td><code>&lt;object&gt;</code>, <code>&lt;prop&gt;</code></td>
<td>Returns value of property</td>
</tr>
<tr>
<td>MakeCustomObject</td>
<td><code>&lt;name&gt;</code></td>
<td>Creates custom object</td>
</tr>
<tr>
<td>IsCustomObject</td>
<td><code>&lt;object&gt;</code></td>
<td>Tests whether object is a custom object.</td>
</tr>
<tr>
<td>DrawObject</td>
<td><code>&lt;obj&gt;</code></td>
<td>Calls the .draw property of an object</td>
</tr>
<tr>
<td>MoveObject</td>
<td><code>&lt;obj&gt;, x, y</code></td>
<td>Calls the .move property of an object</td>
</tr>
<tr>
<td>Clickon</td>
<td><code>obj</code>, <code>[x,y]</code></td>
<td>Calls the .clickOn property of an object</td>
</tr>
</tbody>
</table>

### Misc Event Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetTime</td>
<td><code>&lt;&gt;</code></td>
<td>Gets a number, in milliseconds, representing the time since the PEBL program began running.</td>
</tr>
<tr>
<td>Wait</td>
<td><code>&lt;time&gt;</code></td>
<td>Pauses execution for <code>&lt;time&gt;</code> ms</td>
</tr>
<tr>
<td>IsKeyDown</td>
<td><code>&lt;keyval&gt;</code></td>
<td>Determines whether the key associated with <code>&lt;keyval&gt;</code> is down</td>
</tr>
<tr>
<td>IsKeyUp</td>
<td><code>&lt;keyval&gt;</code></td>
<td>Determines whether the key associated with <code>&lt;keyval&gt;</code> is up</td>
</tr>
<tr>
<td>IsAnyKeyDown</td>
<td><code>&lt;&gt;</code></td>
<td>Determines whether any key is down.</td>
</tr>
<tr>
<td>WaitForKeyDown</td>
<td><code>&lt;keyval&gt;</code></td>
<td>Waits until <code>&lt;keyval&gt;</code> is detected to be in the down state</td>
</tr>
<tr>
<td>WaitForAnyKeyDown</td>
<td><code>&lt;&gt;</code></td>
<td>Waits until any key is detected in down state</td>
</tr>
<tr>
<td>WaitForKeyUp</td>
<td><code>&lt;keyval&gt;</code></td>
<td>Waits until <code>&lt;keyval&gt;</code> is in up state.</td>
</tr>
<tr>
<td>WaitForAllKeysUp</td>
<td><code>&lt;&gt;</code></td>
<td>Waits until all keys are in up state.</td>
</tr>
<tr>
<td>WaitForAnyKeyDownWithTimeout</td>
<td><code>&lt;keyval&gt;</code></td>
<td>Waits until any key is detected in down state with a timeout.</td>
</tr>
</tbody>
</table>
### Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WaitForKeyListDown</strong></td>
<td>&lt;list-of-keyvals&gt;</td>
<td>Waits until one of the keys is in down state</td>
</tr>
<tr>
<td><strong>WaitForKeyPress</strong></td>
<td>&lt;key&gt;</td>
<td>Waits until &lt;key&gt; is pressed</td>
</tr>
<tr>
<td><strong>WaitForAnyKeyPress</strong></td>
<td>&lt;&gt;</td>
<td>Waits until any key is pressed</td>
</tr>
<tr>
<td><strong>WaitForKeyRelease</strong></td>
<td>&lt;key&gt;</td>
<td>Waits until &lt;key&gt; is released</td>
</tr>
<tr>
<td><strong>WaitForListKeyPress</strong></td>
<td>&lt;list-of-keys&gt;</td>
<td>Waits until one of &lt;list-of-keys&gt; is pressed</td>
</tr>
<tr>
<td><strong>WaitForListKeyPressWithTimeout</strong></td>
<td>&lt;list-of-keyvals&gt;</td>
<td>Waits for either a key to be pressed or a time to pass.</td>
</tr>
<tr>
<td><strong>WaitForMouseButton</strong></td>
<td></td>
<td>Waits until any of the mouse buttons is pressed or released, and returns message indicating what happened</td>
</tr>
<tr>
<td><strong>WaitForMouseClickWithTimeout</strong></td>
<td>&lt;timeout&gt;</td>
<td>Waits until any of the mouse buttons is pressed, or a prespecified timeout has elapsed.</td>
</tr>
<tr>
<td><strong>WaitForClickOnTarget</strong></td>
<td>&lt;target&gt;</td>
<td>Waits until any of a set of target objects are clicked.</td>
</tr>
<tr>
<td><strong>WaitForClickOnTargetWithTimeout</strong></td>
<td>&lt;target&gt;, &lt;timeout&gt;</td>
<td>Waits with a max time for a set of targets to be clicked.</td>
</tr>
<tr>
<td><strong>WaitForDownClick</strong></td>
<td></td>
<td>Waits for mouse button to be clicked.</td>
</tr>
<tr>
<td><strong>RegisterEvent</strong></td>
<td>&lt;&gt;</td>
<td>Registers events to trigger based on particular conditions</td>
</tr>
<tr>
<td><strong>StartEventLoop</strong></td>
<td>&lt;&gt;</td>
<td>Starts the event loop</td>
</tr>
<tr>
<td><strong>ClearEventLoop</strong></td>
<td>&lt;&gt;</td>
<td>Clears all trigger events from event loop</td>
</tr>
<tr>
<td><strong>SignalFatalError</strong></td>
<td>&lt;message&gt;</td>
<td>Halts execution, printing out message</td>
</tr>
<tr>
<td><strong>TranslateKeyCode</strong></td>
<td>&lt;&gt;</td>
<td>Converts a keycode to a key name</td>
</tr>
<tr>
<td><strong>TimeStamp</strong></td>
<td></td>
<td>Returns a string containing the current date and time</td>
</tr>
<tr>
<td><strong>GetPEBLVersion</strong></td>
<td>&lt;&gt;</td>
<td>Returns a string indicating which version of PEBL you are using</td>
</tr>
<tr>
<td><strong>GetSystemType</strong></td>
<td>&lt;&gt;</td>
<td>Identifies the type of operating system being used.</td>
</tr>
<tr>
<td><strong>GetVideoModes</strong></td>
<td>&lt;&gt;</td>
<td>Gets list of available screen resolutions</td>
</tr>
<tr>
<td><strong>GetCurrentScreenResolution</strong></td>
<td></td>
<td>Gets the current widthxheight of the screen</td>
</tr>
</tbody>
</table>

55
<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemCall</td>
<td>&lt;command&gt;</td>
<td>Executes command in operating system</td>
</tr>
<tr>
<td></td>
<td>&lt;optional-args&gt;</td>
<td></td>
</tr>
<tr>
<td>LaunchFile</td>
<td>&lt;file&gt;</td>
<td>Launches a file using platform-specific handlers</td>
</tr>
<tr>
<td>GetNIMHDemographics</td>
<td>&lt;code&gt; &lt;window&gt; &lt;file&gt;</td>
<td>Asks NIMH-related questions</td>
</tr>
<tr>
<td>GetSubNum</td>
<td>&lt;window&gt;</td>
<td>Asks user to enter subject number</td>
</tr>
<tr>
<td>MessageBox</td>
<td>&lt;text&gt; &lt;win&gt;</td>
<td>Pops up a message, overtop the entire screen, and waits for a click to continue.</td>
</tr>
<tr>
<td>GetEasyInput</td>
<td>&lt;text&gt; &lt;win&gt;</td>
<td>Gets typed input based on a prompt.</td>
</tr>
<tr>
<td>GetEasyChoice</td>
<td>&lt;text&gt;, &lt;choices&gt;, &lt;output&gt;, &lt;window&gt;</td>
<td>Simple multiple choice</td>
</tr>
<tr>
<td>CountDown</td>
<td>&lt;window&gt;</td>
<td>Displays a 3 2 1 countdown on screen</td>
</tr>
<tr>
<td>IsAudioOut</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is an AudioOut stream</td>
</tr>
<tr>
<td>IsCanvas</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a Canvas</td>
</tr>
<tr>
<td>IsColor</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a Color</td>
</tr>
<tr>
<td>IsFileStream</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a FileStream</td>
</tr>
<tr>
<td>IsFloat</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a floating-point number</td>
</tr>
<tr>
<td>IsFont</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a Font</td>
</tr>
<tr>
<td>IsImage</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is an Image</td>
</tr>
<tr>
<td>IsInteger</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is an integer-type number</td>
</tr>
<tr>
<td>IsLabel</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a Text Label</td>
</tr>
<tr>
<td>IsList</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a List</td>
</tr>
<tr>
<td>IsNumber</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a number</td>
</tr>
<tr>
<td>IsTextBox</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a TextBox</td>
</tr>
<tr>
<td>IsText</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a text string</td>
</tr>
<tr>
<td>IsShape</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is any drawing shape, such as a circle, square or polygon</td>
</tr>
<tr>
<td>IsString</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is a string</td>
</tr>
<tr>
<td>IsWidget</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is any Widget</td>
</tr>
<tr>
<td>IsWindow</td>
<td>&lt;variant&gt;</td>
<td>Tests whether &lt;variant&gt; is any Window</td>
</tr>
</tbody>
</table>

**List Manipulation Functions**
### Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shuffle</td>
<td>&lt;list&gt;</td>
<td>Returns a new list with the items in list shuffled randomly.</td>
</tr>
<tr>
<td>ShuffleRepeat</td>
<td>&lt;list&gt; &lt;times&gt;</td>
<td>Generates a list of n shuffled versions of &lt;list&gt;.</td>
</tr>
<tr>
<td>ShuffleWithoutAdjacents</td>
<td>&lt;nested-list&gt;</td>
<td>Shuffle specifying items that should not appear adjacently</td>
</tr>
<tr>
<td>Repeat</td>
<td>&lt;item&gt; &lt;n&gt;</td>
<td>Repeats an item n times in a list</td>
</tr>
<tr>
<td>RepeatList</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Makes a new list containing the elements of &lt;list&gt; repeated &lt;n&gt; times</td>
</tr>
<tr>
<td>Sequence</td>
<td>&lt;start&gt; &lt;end&gt; &lt;step&gt;</td>
<td>Makes a sequence of numbers from &lt;start&gt; to &lt;end&gt;, with &lt;step&gt;-sized increments</td>
</tr>
<tr>
<td>ChooseN</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Returns a sublist of &lt;n&gt; items from a list, in the order they appear in the original list</td>
</tr>
<tr>
<td>Sample</td>
<td>&lt;list&gt;</td>
<td>Picks a single item randomly from &lt;list&gt;.</td>
</tr>
<tr>
<td>SampleN</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Returns a randomly-ordered sublist of &lt;n&gt; items from a list</td>
</tr>
<tr>
<td>SampleNWithReplacement</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Returns a sublist of &lt;n&gt; items from a list</td>
</tr>
<tr>
<td>DesignLatinSquare</td>
<td>&lt;list1&gt; &lt;list2&gt;</td>
<td>A simple latin square constructor</td>
</tr>
<tr>
<td>LatinSquare</td>
<td>&lt;list&gt;</td>
<td></td>
</tr>
<tr>
<td>DesignGrecoLatinSquare</td>
<td>&lt;list1&gt; &lt;list2&gt; &lt;list3&gt;</td>
<td></td>
</tr>
<tr>
<td>DesignBalancedSampling</td>
<td>&lt;list&gt; &lt;number&gt;</td>
<td></td>
</tr>
<tr>
<td>DesignFullCounterbalance</td>
<td>&lt;list1&gt; &lt;list2&gt;</td>
<td></td>
</tr>
<tr>
<td>CrossFactorWithoutDuplicates</td>
<td>&lt;list&gt;</td>
<td>Returns a list of all pairs of items in the list, excluding pairs that where an element appears twice.</td>
</tr>
<tr>
<td>Rotate</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Rotates a list by &lt;n&gt; items.</td>
</tr>
<tr>
<td>FoldList</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Folds list into length-n sublists.</td>
</tr>
<tr>
<td>Flatten</td>
<td>&lt;list&gt;</td>
<td>Flattens a nested list completely</td>
</tr>
<tr>
<td>FlattenN</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Flattens n levels of a nested list</td>
</tr>
<tr>
<td>Length</td>
<td>&lt;list&gt;</td>
<td>Returns the number of elements in a list.</td>
</tr>
<tr>
<td>First</td>
<td>&lt;list&gt;</td>
<td>Returns the first item in a list.</td>
</tr>
<tr>
<td>Last</td>
<td>&lt;list&gt;</td>
<td>Returns the last item in a list.</td>
</tr>
<tr>
<td>Merge</td>
<td>&lt;list1&gt; &lt;list2&gt;</td>
<td>Combines two lists.</td>
</tr>
<tr>
<td>Append</td>
<td>&lt;list&gt; &lt;item&gt;</td>
<td>Returns new list combining &lt;list&gt; and &lt;item&gt;</td>
</tr>
</tbody>
</table>
## Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PushOnEnd</td>
<td>&lt;list&gt; &lt;item&gt;</td>
<td>Adds &lt;item&gt; to &lt;list&gt; efficiently</td>
</tr>
<tr>
<td>List</td>
<td>&lt;item1&gt; &lt;item2&gt;...</td>
<td>Makes a list out of items</td>
</tr>
<tr>
<td>Sort</td>
<td>&lt;list&gt;</td>
<td>Sorts a list by its values.</td>
</tr>
<tr>
<td>SortBy</td>
<td>&lt;list&gt; &lt;key&gt;</td>
<td>Sorts list by the values in &lt;key&gt;</td>
</tr>
<tr>
<td>Nth</td>
<td>&lt;list&gt; &lt;n&gt;</td>
<td>Returns the nth item in a list.</td>
</tr>
<tr>
<td>Subset</td>
<td>&lt;list&gt; &lt;list-of-indices&gt;</td>
<td>Returns a subset of items from a list</td>
</tr>
<tr>
<td>SetElement</td>
<td>&lt;list&gt; &lt;item&gt; &lt;value&gt;</td>
<td>Sets an element of list to value</td>
</tr>
<tr>
<td>Match</td>
<td>[list],&lt;item&gt;</td>
<td>Returns a list of 0/1s, indicating which elements of list match item.</td>
</tr>
<tr>
<td>Filter</td>
<td>[list],&lt;indicators&gt;</td>
<td>Filters a list based on a 0/1 list produced by Match.</td>
</tr>
<tr>
<td>Levels</td>
<td>[list]</td>
<td>Returns a sorted list of unique elements in list.</td>
</tr>
<tr>
<td>Rest</td>
<td>&lt;list&gt;</td>
<td>Returns a list minus its first element</td>
</tr>
<tr>
<td>ExtractListItems</td>
<td>&lt;list&gt; &lt;list-of-indices&gt;</td>
<td>Gets a subset of items from a list</td>
</tr>
<tr>
<td>IsMember</td>
<td>&lt;item&gt; &lt;list&gt; &lt;list-of-indices&gt;</td>
<td>Checks whether &lt;item&gt; is a member of &lt;list&gt;</td>
</tr>
<tr>
<td>Replace</td>
<td>&lt;template&gt; &lt;replacementList&gt;</td>
<td>Replaces items in a data structure</td>
</tr>
<tr>
<td>Lookup</td>
<td>&lt;key&gt; &lt;keylist&gt; &lt;database&gt;</td>
<td>returns element in &lt;database&gt; corresponding to element of &lt;keylist&gt; that matches &lt;key&gt;.</td>
</tr>
<tr>
<td>Transpose</td>
<td>&lt;list-of-lists&gt;</td>
<td>Transposes a list of equal-length lists.</td>
</tr>
<tr>
<td>SubList</td>
<td>&lt;list&gt; &lt;start&gt; &lt;finish&gt;</td>
<td>Returns a sublist of a list.</td>
</tr>
<tr>
<td>RemoveSubset</td>
<td>&lt;list&gt; &lt;list-of-pos&gt;</td>
<td>Removes items at positions &lt;list-of-pos&gt; from a list.</td>
</tr>
<tr>
<td>ListToString</td>
<td>&lt;list&gt;</td>
<td>Concatenates all elements of a list into a single string</td>
</tr>
<tr>
<td>Insert</td>
<td>&lt;list&gt;,&lt;item&gt;,&lt;pos&gt;</td>
<td>Inserts &lt;item&gt; into &lt;list&gt; at &lt;pos&gt;</td>
</tr>
<tr>
<td>ListBy</td>
<td>&lt;list&gt;,&lt;conds&gt;</td>
<td>Segments a list into sublist by the values of a second list</td>
</tr>
</tbody>
</table>

### String Management Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>&lt;num&gt;</td>
<td>Returns string with &lt;num&gt; linefeeds.</td>
</tr>
<tr>
<td>Tab</td>
<td>&lt;num&gt;</td>
<td>Returns string with &lt;num&gt; tabs.</td>
</tr>
</tbody>
</table>
### Chapter 5. Function Quick Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td><code>&lt;value&gt; &lt;num&gt;</code></td>
<td>Makes string from value exactly <code>&lt;num&gt;</code> characters by truncating or padding.</td>
</tr>
<tr>
<td>Enquote</td>
<td><code>&lt;text&gt;</code></td>
<td>Returns string surrounded by quote marks.</td>
</tr>
<tr>
<td>Uppercase</td>
<td><code>&lt;string&gt;</code></td>
<td>Returns uppercased string</td>
</tr>
<tr>
<td>Lowercase</td>
<td><code>&lt;string&gt;</code></td>
<td>Returns lowercased string</td>
</tr>
<tr>
<td>ReplaceChar</td>
<td><code>&lt;string&gt; &lt;char&gt;&lt;char2&gt;</code></td>
<td>Substitutes <code>&lt;char2&gt;</code> for <code>&lt;char&gt;</code> in <code>&lt;string&gt;</code>.</td>
</tr>
<tr>
<td>SplitString</td>
<td><code>&lt;string&gt; &lt;split&gt;</code></td>
<td>Splits <code>&lt;string&gt;</code> into a list of <code>&lt;split&gt;</code>-delimited substrings.</td>
</tr>
<tr>
<td>SplitStringSlow</td>
<td><code>&lt;string&gt; &lt;split&gt;</code></td>
<td>Splits <code>&lt;string&gt;</code> into a list of <code>&lt;split&gt;</code>-delimited substrings.</td>
</tr>
<tr>
<td>StringLength</td>
<td><code>&lt;string&gt;</code></td>
<td>Returns the length of a string</td>
</tr>
<tr>
<td>SubString</td>
<td><code>&lt;string&gt; &lt;position&gt; &lt;length&gt;</code></td>
<td>Returns a substring</td>
</tr>
<tr>
<td>FindInString</td>
<td><code>&lt;string&gt; &lt;key&gt;&lt;pos&gt;</code></td>
<td>Returns position of <code>&lt;key&gt;</code> in <code>&lt;string&gt;</code>, starting at position <code>&lt;pos&gt;</code>.</td>
</tr>
<tr>
<td>StripSpace</td>
<td><code>&lt;string&gt;</code></td>
<td>Strips whitespace from the start and end of <code>&lt;string&gt;</code>.</td>
</tr>
<tr>
<td>StripQuotes</td>
<td><code>&lt;string&gt;</code></td>
<td>Strips quotation marks from the start and end of <code>&lt;string&gt;</code>.</td>
</tr>
</tbody>
</table>
Chapter 6

The PEBL Launcher

The PEBL Launcher is the best way to navigate and launch PEBL experiments, especially for novices or research assistants. It allows one to specify a few specific options that are frequently changed, navigate through the PEBL Test Battery, and create and save ‘experiment chains’ to let you run multiple experiments in a row.

Figure 6.1: Screenshot of PEBL Launcher.
6.1 History of the Launcher
Prior to 2011, a front-end launcher was only available for PEBL on Windows. It was written in Visual Basic 6, which was old-fashioned, single-platform, no longer supported by Microsoft, and created a situation where a critical piece of PEBL infrastructure depended on a non-free tool. The main obstacle to a new launcher has always been: PEBL needs a cross-platform launcher using a free software, and we don’t want to have to distribute a whole additional interpreter. This means that Python, wxBasic, TCL/TK, etc. were out of the consideration. Why couldn’t there be an easy-to-use cross-platform programming tool we could use?

As of PEBL Version 0.12, we found one: PEBL itself. PEBL is not really designed to create GUI applications, but it can be beat into submission to do so. For Version 0.12, enough filesystem access functions and other features were available to make a reasonable launcher.

For PEBL 0.14, the launcher received a major overhaul. With the advent of custom objects, we added a bunch of GUI objects (buttons, scrolling text boxes, checkboxes, menus, etc.) that enabled a much more polished version of the launcher that integrates better with other desktop options. This allowed streamlining the launcher, adding functionality, improving its usefulness in the research lab. This includes the ability to set and change script-based parameters, which allows an experimenter to better tailor the PEBL battery tests to their particular needs.

6.2 How it works
The simplest usage of the Launcher is that you use the file selector on the left to choose a .pbl file, then click the button "Run selected script" to run that experiment. ONLY .pbl files and directories will appear in the file window.
Chapter 6. The PEBL Launcher

6.3 Features

6.3.1 File browser

On the left is a file browser. It will only show .pbl files and subdirectories. To navigate to a subdirectory, simply click on the directory to select it, then click on the selected directory. To move back up a directory, click on the ' '..\' row. When you have a .pbl file selected, you can use the 'Run selected script' button to launch it.

6.3.2 Participant code

This will allow you to select the participant code you want sent to any experiments you are about to run. By default, PEBL saves the last experiment code when you exit, and then reloads it the next time, incrementing by one. This makes it easier to avoid colliding participant codes and overwriting data. Participant code need not be a number, but the launcher currently does not understand how to increment non-numeric codes, and will probably restart at 1. The plus button next to the code box will increment the current number by 1, which is useful if you are running multiple sessions in a row.

The automatic incrementation of participant code can be turned off by opening the fileselct.pbl file and changing the variable gAutoSubcode from 1 to 0. When an experiment is launched, the specified code will be fed into the experiment using the -s command-line option, and will be bound to the gSubNum variable. Some of the standard experiments will ask you to enter a participant code regardless of whether you have one selected. If that is the case, you should be able to edit the script to remove the request to specify a participant code. However, most experiments in the test battery should only ask the experimenter to specify a participant code if the participant code is '0', which is what it will be when no -s command is given. So, if you are using code 0, many of the experiments will ask you to enter a code after they launch.

6.3.3 Experimenter code

Many times, you may wish to keep track of the experimenter or research assistant who collected the data. Have them enter their name in the 'experimenter'
The PEBL Launcher window. The name will be saved on exit. The experimenter code will be saved to the runlog file (see below).

6.3.4 Language

Some experiments have instructions and stimuli that are translated into different languages. Enter your two-character language code in the language box to tell the experiment what language to use. If your chosen language is not available, the experiment will fall back to English. For Chinese and related languages, setting this will also change the default fontface used. If you want to translate an experiment into your own language, ask on the PEBL mailing list.

6.3.5 Command Line Options

There are a number of command line options available for PEBL that are not present as options in the launcher. If you want to use any, you can type them in the “Command line Options” box and the launcher will pass them to PEBL. You can use these to specify -V options that pass parameters into your experiment (e.g., controlling whether a practice or a test round is given).

6.3.6 Edit and Parameters

The Edit button will let you edit the parameters used in the test. When you edit and save the parameter set, it will then appear as an option in the parameters pulldown. Before you run an experiment, you can select the parameter set you want to use from the pulldown (or save it permanently to an experiment chain).

6.3.7 Fullscreen Mode

If you want to launch your experiment in full-screen mode to improve video latency and to avoid distractions, check this box. The secret escape key combo is ctrl-alt-shift-\: hit these four to abort out of an experiment before it is complete.

6.3.8 Demographics Collection

The U.S. NIMH requires a number of demographic variables for research they fund. Checking this box will collect this data and save it to a data log file called demographics-log.csv, prior to running your experiment or experiment chain.

6.3.9 Experiment Chains

The launcher allows you to set up a ‘chain’ of experiments that get run in sequence. All the experiments will be run consecutively, with an identical subject code. This is accomplished by running a separate instance of PEBL for each experiment. This can sometimes lead to a ‘flash’ between each experiment if running in fullscreen mode. Below the experiment chain window is a pulldown
that lets you select the particular chain you want to use. The default chain is loaded by default, and is also responsible for setting the parameter sets above.

6.3.10 Saving Experiment Chains

When you exit the launcher, the current experiment chain will get saved in the current config file. By default, this file is called default.config. This same file is loaded when the launcher starts again, restoring your settings. By hitting the 'save chain' button, you will be asked to enter a new name to save the current configuration under. Similarly, 'Delete chain' will delete the file in which the current chain is saved. A chain can be loaded at start-up (by specifying the name of the config file with the -v command-line option).

6.3.11 Editing Experiment chains

A chain can be edited by inserting, appending, or deleting steps, or clearing the entire chain, using the buttons below the experiment chain box. Be sure to save the chain after editing so your edits will be saved.

6.3.12 Loading Experiment Chains

A previously saved experiment chain can be loaded by selecting the chain name from the pulldown selection box.

6.3.13 Description and Screenshot

On the right side of the launcher is a window that will show a screenshot and print a description of a script when it is highlighted. These need to be created by hand for each script. The launcher does its best to show you a preview of the test inside any directory. But to run an experiment, you need to select a .pbl file in the file window on the left. So, even if a screenshot appears on the right, you need to select the actual .pbl file to run the experiment.

6.3.14 Message feedback windows

Whenever a PEbl script runs, error and debug messages are saved to files called stdout.txt and stderr.txt, within the directory the file is run from. When a test is completed, PEbl will look for and try to load these files in the tabbed window at the bottom of the launcher. stdout.txt typically contains any messages saved using the Print() command, and is useful for debugging code. If an experiment crashes, it will be logged in the stderr.txt file and the Error messages tab. In addition, a lot of bookkeeping information is saved to that file, which can help diagnose other possible problems. If you need to access these files directly to help report bugs, you can open them using the 'Open Debug Output' and 'Open error output' buttons.
6.3.15 Other buttons

The launcher has a number of other buttons to help you use PEBL. These include:

- **Open/Edit selected** On the lower left, there is a button labeled “Open/Edit selected”. This will open a selected .pbl script in a text editor, and will open a directory in your system’s file manager. An easy way to look at or make changes to the script, or to locate data files after a script is run.

- **Wiki** This will launch a web browser that will take you to the PEBL wiki page that is related to the currently selected test. This should help provide information about the test, its background, parameters, and data format.

- **Combine data** This will launch a data combining utility described below. It will help merge all your data files together into a single spreadsheet.

- **Open debug output** Whenever an experiment is run, any time you use the Print() function, it will print the resulting text to a file names stdout.txt in the directory it was run in. This button will open that file.

- **Open error output** Whenever an experiment is run, the error and status messages are saved to a file called stderr.txt in the directory it was run in. This button will open that file.

6.3.16 Menu

A lot of functionality is present in menus at the top of the window.

- **File/Exit** This will exit the launcher and save the current configuration options to the named experiment chain.

- **Options/change launcher size**. The launcher has trouble running on netbooks with only 600 pixels vertical distance. This will make a 'small' launcher that is more compact, but only works next time the launcher is opened.

- **Help/About** This provides a short description of the launcher.

- **Help/Manual** This opens the PEBL .pdf manual. The manual is located in different places on each platform, and will change names for each release.

- **Help/Website** This will take you to the main PEBL website.

- **Help/Wiki** This button will take you to the PEBL wiki, and do its best to find a WIKI page related to the experiment you are looking at. They won’t always exist, and if not, you can always sign in and make your own.

- **Help/Tutorial** This will open a wiki page containing a basic PEBL usage tutorial.
Chapter 6. The PEBL Launcher

- Help/Review This will let you provide feedback about PEBL
- Help/Donate This will let you donate funds to help support PEBL development.

6.4 Launching an experiment

To launch an experiment, navigate through the directories in the file listing box. Only directories and files with the .pbl extension are shown in this box. To open a directory, click once to move the highlight box onto the directory name, and a second time to open the directory. When a new directory is opened, the first available .pbl file will be automatically selected. To run that script, just press the 'Run selected script' button above the file select box. It will run with the specified parameter, including subject code, language, fullscreen mode. In addition, if the 'collect demographics' button is selected, a demographic survey will happen prior to the study running.

6.5 Launching an experiment chain

If you have a series of experiments you want to run, create an experiment chain and launch it using the 'Launch chain' button above the experiment chain selection box. Tip: Use experiment chains even if you are running just single experiment, with just a single experiment selected. This gives faster access and is less error-proned.

6.6 Translating the Launcher

You can translate the launcher to your own language. Open the launcher file (fileselect.pbl), and go to the end of the script, to a function named "GetStrings":

```plaintext
define GetStrings(lang)
{
    lang <- Uppercase(lang)
    if(lang == "EN")
    {
        gRunText <- "Run selected script"
        gOpenText <- "Open"
        gExitText <- "EXIT"
        gViewDebugText <- "View debug output"
        gViewErrorText <- "View error output"
        gAddToChainText <- "Add to Chain"
        gClearChainText <- "Clear Chain"
        gSaveChainText <- "Save Chain"
    }
    ....
```

The labels used in the launcher all appear here. You should be able to just translate the text of each one into the language of your choice. Send the translations back to the author so they can be incorporated into the next launcher.
version. You can also make a section in the if statement for your particular language. When you change the language in launcher, it will save that option and use your language of choice next time.

### 6.7 Utility: Parameter setting

Version 0.14 of the PEBL Launcher allows you to set parameters from tests before launching. A basic screenshot is shown below:

![Parameter Setting Screenshot](image)

Here, the first column shows the name of the parameter. The second is the current value, which can be edited by clicking on the box and typing a new value. When complete, hit enter and the new set will be recorded. The rightmost column provides a basic description of the parameter, and its default value.

To create a new parameter set, write the name of the parameter set you want to use in the box at the top of the screen. Then, when you hit 'Save file and exit', it will be saved to this parameter file. There is no need to include .par in the filename, as it will be added if you do not add it yourself. To edit a current parameter set, select the parameter set you want and press 'edit' in the main window.

### 6.8 Utility: Combining data files

Once an experiment is done, the data files are typically stored within the data directory in which the test appeared. Furthermore, each participant may be saved in his or her own subdirectory. On some tests, a merged or pooled data file is also saved, but this is not always the case. In order to merge all of you data into one master file, you can use PEBL’s data combining tool, accessible through a button on the lower left of the launcher.

To use this, navigate to the data directory of your tests, and click the 'Combine data' button. A screen like the one below should open.
Chapter 6. The PEBL Launcher

On the upper right, a list of all files within the selected directory (and subdirectories) will be displayed. You want to choose some (but probably not all) of these. You can choose a subset by typing match values into the match and exclude boxes on the left. Currently, the * indicates all files will match. You may want to just include .csv files, in which case deleting the * and typing csv into the box will bring up only files with csv in their name. You may want to exclude summary files, in which case you can type summary in the excludes box.

Each time you change the selection criteria and hit enter, the list of files will update. A preview of any of the files can be seen in the lower right window.

In the match and exclude boxes, spaces act as logical 'or's. matching with the following '* csv' will match all files, because the first * will match all files. Or can also be specified using the | character. The & character can be used to specify AND criteria. To match csv files from participant 300, you would enter '300&csv'. The matches are processed before the excludes.

Once you have selected the right files using match and exclude criteria, you should determine whether the files have a header. If they have a header, you probably want to remove the header, including just once in the merged file. The combiner is not smart enough to detect this on its own, so you must check 'files contain header' if you want the header stripped from each individual data set and added to the final merged set.

Finally, especially if your data does not have a participant code in it, you may check 'add filename to data' which will add a column at the beginning of the data indicating the source file of each row of data.

Once you are ready, you can choose 'combine and save' which will save the data to the filename you specified in 'save file'. If you use the combiner more than
once, be sure to exclude 'pooled' from your match list so you don’t get multiple copies of your data. You can also choose 'combine and open', which will create the pooled file, but then try to open in with whatever program is associated with that file type (i.e., microsoft excel for .csv files).
Chapter 7

The PEBL Psychological Test Battery

This chapter contributed by Bryan Rowley in collaboration with Shane Mueller

About the Battery
This site is for a battery of psychological tests implemented in PEBL and distributed (and redistributable) freely. They are designed to be easily used on multiple computing platforms, running natively under Win32, Linux, and OSX Operating Systems. The tests are designed to implement a wide range of computer-administered psychological tests and experiments of interest to neuropsychological, cognitive, clinical communities.

The current version of the battery is designed to work with PEBL version 0.14 and was released in 2014. It is distributed with PEBL 0.14, and is automatically installed in My Documents\pebl.0.14\battery on windows.

These tests are designed to implement a wide range of tests that are used throughout the psychological, neuropsychological research and clinical communities. Some are reimplementations of tests that are only available on limited computing platforms or cost hundreds of dollars. Each experiment saves the complete data set for later analysis, and many compute basic analyses that it writes in report format.

Tests
The following table describes the basic tests currently implemented in the PEBL Test Battery. Many of them represent the only Free version of proprietary tests available anywhere. They include a free Iowa Gambling Task, a free version of the TOVA®, a free Wisconsin Card Sort Test®, a free version of Conners Continuous Performance task, and a number of other useful tasks, with more to come. All screenshots found on this page are released into the public domain, and can be used for whatever purpose without copyright assignment, including in academic papers. More information on tests is found in the PEBL WIKI
### Chapter 7. The PEBL Psychological Test Battery

**Table 7.1: Test Battery**

<table>
<thead>
<tr>
<th>PEBL Test/Version of:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bechera's Gambling Task (version of Bechara's Iowa Gambling Task [wikipedia])</td>
<td>Choose from four decks, each choice with a cost and each providing reward. Used for tests of executive control. <strong>Key Skills used:</strong> Decision Making, Strategy and Problem Solving, Risk Assessment. <strong>Note:</strong> the task requires individuals to decide on which deck to choose from, with the chance of losing in the process. Test can be modified to ask individual to achieve a certain amount of money.</td>
</tr>
<tr>
<td>The &quot;Hungry Donkey&quot; Task A version of Bechera's Gambling Task for children</td>
<td>The donkey chooses from four doors, each door has a cost and reward in apples. Used for tests of executive control. <strong>Key Skills used:</strong> Fine-motor skills, Visual processing. <strong>Note:</strong> Test can be modified to ask individual to reach a certain number of apples (i.e. 10 apples) in a certain amount of time.</td>
</tr>
<tr>
<td>TOAV: Test of Attentional Vigilance A Version of TOVA®: Test of Variables of Attention [wikipedia]</td>
<td>22-minute test requiring subject to detect a rare visual stimulus (top or bottom). Used to diagnose ADD, ADHD, etc. <strong>Key Skills used:</strong> Concentration, Reaction Time, Attention <strong>Note:</strong> This task requires the individual to concentrate for an extended period of time. Thus, the extent to which their reaction time scores alter through the duration of this test can be indicative of how their attention levels have been affected.</td>
</tr>
</tbody>
</table>
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of:</th>
<th>Description</th>
</tr>
</thead>
</table>
| **PEBL Continuous Performance Test**<br>Version of Conners CPT wikipedia | 14-minute vigilance test requiring subject respond to non-matches. Used to diagnose ADD, ADHD, etc.  
**Key Skills Used:** Reaction Time, Attention, Concentration.  
**Note:** The test length allows for observation of how their results change overtime (i.e. attention levels altering). |
| **PEBL Perceptual Vigilance Task (PPVT)**<br>Wilkinson & Houghton's Psychomotor Vigilance Task wikipedia | A vigilance task used to detect vigilance and sleep lapses.  
**Key Skills Used:** Reaction Time, Attention, Concentration.  
**Note:** The individual’s results can be viewed in data section, and we can observe how their performance declines or improves throughout test duration. |
| **Berg's Card Sorting Test**<br>version of Berg's (1948) Wisconsin Card Sorting Test wikipedia | Sort multi-attribute cards into piles according to an unknown and changing rule.  
**Key Skills used:** Strategy and Problem Solving, Decision Making, Inhibition, Working Memory.  
**Note:** The results from the data section provide an indication of which rule (shape, color or number) is easiest for the individual via reaction time. We are able to see how the individual’s working memory is operating by their ability to recall which rule is active (via correct responses). |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of:</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Simple Response Time**  
[Link](https://en.wikipedia.org/wiki/Simple_response_time) | Detect the presence of a visual stimulus, as quickly and accurately as possible.  
**Key Skills Used:** Reaction Time, Attention, Fine Motor Skills.  
**Note:** The task allows for observation of how their attention and reactivity alter throughout the test’s duration. The individual can also work on their executive control and fine motor ability. |
| **Digit Span**  
**Key Skills Used:** Working Memory, Numerical Processing, Short Term Memory.  
**Note:** Primacy, Recency effects can be observed in this task (i.e. which numbers in the set are being remembered, first numbers or last numbers). |
| **Partial Report Procedure**  
**Key Skills Used:** Reaction Time, Decision Making, Working Memory.  
**Note:** Individuals are required to make quick decisions based on a brief stimulus shown. Not recommended for people with slow reaction times. |

74
<table>
<thead>
<tr>
<th>PEBL Test / Version of</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Implicit Association Test</strong>&lt;br&gt;A test of automatic associations between memory representations. <a href="https://en.wikipedia.org/wiki/Implicit_Association_Test">wikipedia</a></td>
<td>Tests associations between two sets of binary classifications.</td>
</tr>
</tbody>
</table>
**Key Skills Used:** Strategy and Problem Solving, Color Processing, Hand-eye coordination, Fine Motor Skills.  
**Note:** Test cannot be completed successfully for color-blind individuals. Task is great for individuals trying to improve on executive control, and requires both strategy and problem solving skills to complete successfully. |
| **Symbol Counter Task**<br>Garavan (2000) counter task | Useful indicator of executive control.  
**Key Skills used:** Reaction Time, Working Memory, Selective Attention.  
**Note:** We can view if the individual will be able to recall which symbols are associated with which shift tab (i.e. a measure of working memory via correct responses). |
Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Four choice response time</strong>&lt;br&gt;Wilkinson &amp; Houghton's 4-choice response time wikipedia</td>
<td>Respond to a plus sign that appears in one of four corners of the screen. <strong>Key Skills used:</strong> Reaction Time, Selective Attention, Visual Processing. <strong>Note:</strong> the task measures how quickly the individual’s attention leads them to the correct location, combining visual processing abilities with reaction time.</td>
</tr>
</tbody>
</table>

| **Time Wall**<br>UTCPAB's Time wall | Estimate the time when a moving target will reach a location behind a wall. **Key Skills Used:** Reasoning, Calculating, Reaction Time, Strategy and Problem Solving. **Note:** this task requires tracking of an object after its disappearance. It requires the individual to in a sense to imagine the location of this object using precise calculating (of object’s speed). |

| **PEBL Compensatory Tracker**<br>Similar to Maleig & Jolley's CompTrack | Use mouse/trackball to keep a randomly moving target inside a bullseye. **Key Skills used:** Fine Motor Skills, Strategy and Problem Solving, Hand Eye Coordination. **Note:** this task can be helpful for individuals wanting to get better with using a mouse for the computer. |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test / Version of:</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Lexical Decision**   | Determine whether a stimulus is a word or nonword.  
**Key skills used:** Linguistic Processing, Language Processing.  
**Note:** the words are able to be changed for the test. They can be changed to fit closely to an individual's expertise (ex. If individual is aphasic but has an interest in bands, the words can be altered to include words of bands they listen to frequently). |
| **Mental Rotation**    | Determine whether two figures are identical, subject to rotation.  
**Key Skills used:** Reasoning, Visual Processing, Decision Making.  
**Note:** while observing both objects, the individual is required to make a decision of whether the objects are similar, and requires precise reasoning due to their similarities (i.e. be able to reason that object on left looks identical to the object on the right, only inverted from the object on the right). |
| **Matrix Rotation**    | Determine whether a 6x6 matrix is the same (with rotation) as another.  
**Key skills used:** Selective Attention, Working Memory, Visual Processing.  
**Note:** Working Memory is being tested, we can see how individual's object manipulation or 'visuo-spatial sketchpad' is operating (i.e. correct responses being a measure of working memory, and the 'sketchpad' the specific component being measured). |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of:</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Spatial Cueing** | Given a probabilistic cue of where a stimulus will appear, respond as fast as possible.  
Posner's attentional cueing (spotlight) task. Wikipedia  
*Key Skills used:* Selective Attention, Inhibition.  
*Note:* this task tests the individual’s ability to make the correct response regardless of the correct cue or the distracter cue. We can view how the distracter cue affects the individual via correct responses and reaction time. |
| **Two column addition** | Add three two-digit numbers and respond quickly and accurately.  
UTC test battery's 2-column addition.  
*Key Skills Used:* Mathematical Processing, Numerical Processing, Working Memory.  
*Note:* it is important to distinguish between Mathematical and Numerical, as mathematical processing in this test refers to the manipulation of numerical information, whereas numerical processing refers to the knowledge of numerical information (i.e. the understanding that the number ‘one’ means ‘1’.) |
| **Stroop task** | Respond to either the color or name of stimuli.  
Stroop's attention task. Wikipedia  
*Key Skills Used:* Inhibition, Selective Attention.  
*Note:* Reaction Time is recorded in the data section, allowing for analysis of which trails are easiest, and which are most challenging. |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test / Version of:</th>
<th>Description</th>
</tr>
</thead>
</table>
| **PEBL Manual Dexterity** | Move a noisy cursor to the target.  
**Key Skills used:** Fine Motor Skills, Strategy and Problem Solving, Hand-eye Coordination.  
**Note:** This task is helpful for individuals trying to improve their mouse ability with the computer. |
| **PEBL Trail-making test** | Connect the dots task.  
**Key Skills used:** Language Processing, Numerical Processing, Hand-eye coordination.  
**Note:** This task tests both linguistic and numerical processing, and tests the individual’s ability to navigate to the correct location (i.e. visual processing). |
| **Aimed Movement (Fitts’s Law) test** | Mouse-driven implementation of classic perceptual-motor task.  
**Key Skills used:** Hand-eye coordination, Fine Motor Skills, Concentration.  
**Note:** The number of trails (105) requires continuous concentration on the participants’ behalf. |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of:</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Pursuit Rotor task**  
Classic mechanical test device | Mouse-driven motor pursuit.  
**Key Skills used:** Hand–eye coordination, Fine Motor Skills, Strategy and Problem Solving.  
**Note:** The task requires the individual to adapt to the rate at which the circle is moving, thus requiring incorporation of a calculating strategy to complete successfully. |
| **Match to sample task**  
Classic non-visual short-memory task | Match a matrix pattern to one presented after a delay.  
**Key Skills used:** Reasoning, Calculating, Color-processing.  
**Note:** color-blind individuals will not be as successful in this task. |
| **Corsi block test**  
Version of physical "Corsi block-tapping test" | Measure of visual-spatial working memory.  
**Key Skills used:** Working Memory, Visual Processing.  
**Note:** reaction time can be measured in the trails varying in length. |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test / Version of:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Detection test</td>
<td>Assess whether participant sees change in a display of colored circles.</td>
</tr>
<tr>
<td>Version of numerous change blindness paradigms</td>
<td><strong>Key Skills used:</strong> Selective attention, Visual processing, Concentration.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> the changing object may not be so obvious at first, so additional concentration may be required.</td>
</tr>
</tbody>
</table>

| Clock Test                    | Watch a clock, and respond whenever it skips a beat.                       |
|                               | **Key Skills used:** Selective attention.                                  |
| Mackworth's Sustained attention test | **Note:** Reaction Time is revealed in the data section, indicating the individual's attention levels as the task progresses. |

<p>| Device Mimicry Test           | Operate a 4-df etch-a-sketch to recreate paths produced by the computer. |
|                               | <strong>Key Skills used:</strong> Calculating, Hand-eye coordination, concentration, Fine Motor Skills, Strategy and Problem Solving. |
|                               | <strong>Note:</strong> This task requires precision to complete successfully. Test can be very helpful for individual's trying to improve their computer skills, or in cognitive rehabilitation sessions. |</p>
<table>
<thead>
<tr>
<th>PEBL Test/Version of</th>
<th>Description</th>
</tr>
</thead>
</table>
| Item-Order Test      | Assess two consecutive letter strings, and determine whether they are the same or different. Different trials are creating either by changing identity of a letter or the order of two adjacent letters.  
**Key Skills used:** Language Processing, Working Memory.  
**Note:** Does the duration of the test result in better or poorer performance? This can be measured in the data section. |
| Letter-Digit substitution  
Version of UTCPAB and Wechsler tests | Recode stimuli according to a letter-digit code chart.  
**Key Skills used:** Language Processing, Numerical Processing.  
**Note:** Great test to use with Aphasic patients to see how they map language information with mathematical information. Reaction time revealed in data section. |
| Math Processing | Do simple arithmetic problems.  
**Key Skills used:** Mathematical processing, Numerical processing, Reaction Time.  
**Note:** Important to distinguish between mathematical and numerical processes, as the former refers to the manipulation of numerical information, and the latter refers to basic processing of numerical information (i.e. that ‘1’ means ‘one’). |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test /Version of:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Span (Visual)</strong></td>
<td>See a sequence of items, then respond using mouse or touchscreen.</td>
</tr>
<tr>
<td><strong>Classic experimental paradigm</strong></td>
<td><strong>Key Skills used:</strong> Working Memory, Short Term Memory, Visual Processing. <strong>Note:</strong> Individuals familiarity with certain objects may result in better recall for those objects (i.e. animal lovers).</td>
</tr>
</tbody>
</table>

| **Object Judgment** | Determine whether two polygons are identical, while manipulating shape, orientation, size. **Key Skills used:** Calculating, Reasoning, Visual Processing. **Note:** may require concentration due to the duration of task. Task requires visual manipulation of the stimuli presented. |

| **Pattern Comparison Test** | Examine two grid patterns and determine whether they are the same. **Key Skills used:** Calculating, Visual Processing. **Note:** pattern-samediff.pbl requires reaction time (found in data section), while pattern-sequential.pbl requires working memory to function (via correct responses). |
### Probability Monitor

Watch a set of gauges to determine when one gets a hit.

**Key Skills used:** Calculating, Inhibition, Visual Processing, Reasoning.

**Note:** while trying to detect a pattern (calculating and reasoning), the individual is required to inhibit other random dials on later trails (trails 2 and 3). Reaction time is measured in data section.

### Situation Awareness Test

Watch a set of moving targets and respond to probes about their locations and identities.

**Key Skills used:** Selective Attention, Working Memory, Visual Processing.

**Notes:** Test great for combining visual awareness with working memory.

### Comfort scales

Respond to four visual-analytic scales about different dimensions of comfort.

**Key Skills used:** Linguistic Processing, Calculating.

**Note:** Allows for extensive self reflection, and requires linguistic ability for responses (to indicate how they feel).
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test / Version of:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed tapping test</strong></td>
<td>Tap a key as quickly as possible. <strong>Key Skills used:</strong> Fine Motor Skills. <strong>Note:</strong> can be used for individuals in rehabilitation sessions.</td>
</tr>
<tr>
<td><strong>Time tapping test</strong></td>
<td>Tap for a production period at a prespecified pace. <strong>Key Skills used:</strong> Calculating, Working Memory. <strong>Note:</strong> requires individual to recall and implement the pace at which they are required to tap.</td>
</tr>
<tr>
<td><strong>Tower of Hanoi test</strong></td>
<td>Solve game with disks. <strong>Key Skills used:</strong> Calculating, Reasoning, Hand-eye coordination, Fine Motor Skills, Working Memory, Visual Processing, Strategy and Problem Solving. <strong>Note:</strong> Able to track the individual’s number of moves. Task is very great for a multitude of cognitive abilities, and is helpful for patients with cognitive disorders.</td>
</tr>
</tbody>
</table>
### The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Two-column addition** | Do mental arithmetic of at least three two-digit summands.  
**Key Skills used:** Mathematical Processing, Working Memory, Calculating.  
**Note:** Individual can be asked how they decided to solve the problems (i.e. with what strategy: first column then the next two columns, or adding all the numbers at once etc.) |
| **Visual Search** | Find the target amidst clutter.  
**Key Skills used:** Language Processing, Visual Processing, Selective Attention, Colour Processing, Inhibition, Concentration.  
**Note:** X’s and O’s are quite distinguishable letters. O’s look more similar to the other letters than X does, and therefore the trials with X’s and O’s can be compared to see which ones are easier (via correct response or not) and found quicker (via reaction time). |
| **Attentional Network Task**  
*Version of Fan et al.’s ANT* | Assess three types of attention.  
**Key Skills used:** Selective Attention, Reaction Time, Inhibition.  
**Note:** The data section reveals trial and the corresponding reaction times. Can be viewed is how their attention processes alter through the test’s duration. |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test / Version of:</th>
<th>Description</th>
</tr>
</thead>
</table>
| **PEBL Balloon Analog Risk Task**  
Version of LeJuez et al's BART | Assess three types of attention.  
**Key Skills used:** Risk Assessment and risk aversion.  
**Note:** Test can be modified to ask the participant to reach a certain money value in a set amount of time. |
| **Dot Judgment Task**  
Determine which field has more dots. | **Key Skills used:** Calculating, Decision Making  
**Note:** Threshold provides an opportunity to observe how the individual performs (with correct judgment) when dot amounts are similar. |
| **Flanker Task**  
Eriksen's Flanker Task | Make direction response with distraction.  
**Key Skills used:** Selective Attention, Reaction Time, Inhibition.  
**Note:** The data section reveals trial and the corresponding reaction times. Can be viewed is how their attention processes progress through the test’s duration. |
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Go/No-go Task</strong></td>
<td>Classic continuous performance task. <strong>Key Skills used:</strong> Inhibition, Reaction Time, Language Processing, Selective attention. <strong>Note:</strong> The data section allows for observation of their scores, and to view if their inhibition skills are increasing or decreasing with time.</td>
</tr>
<tr>
<td>Version of Bezdjian's 2009 Implementation</td>
<td></td>
</tr>
</tbody>
</table>

![Go/No-go Task](image1.png)

<table>
<thead>
<tr>
<th>Manikin Task</th>
<th>Assess mental rotation.</th>
</tr>
</thead>
</table>

![Manikin Task](image2.png)

<table>
<thead>
<tr>
<th>TLX Workload Assessment</th>
<th>Assess workload of task on multiple dimensions. <strong>Key Skills used:</strong> Concentration, Linguistic Processing, Calculating. <strong>Note:</strong> requires the individual to self reflect, read the information, and calculate their levels according to the scale provided.</th>
</tr>
</thead>
<tbody>
<tr>
<td>An implementation of NASA's TLX workload assessment <a href="https://en.wikipedia.org/wiki/TLX_workload_assessment">wikipedia</a></td>
<td></td>
</tr>
</tbody>
</table>

![TLX Workload Assessment](image3.png)
### Chapter 7. The PEBL Psychological Test Battery

<table>
<thead>
<tr>
<th>PEBL Test/Version of:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Muller-Lyer Illusion</strong></td>
<td>Psychometric study of Illusion.</td>
</tr>
<tr>
<td>Classic perceptual illusion</td>
<td><strong>Key Skills used:</strong> Calculating, Reaction Time.</td>
</tr>
<tr>
<td><a href="https://en.wikipedia.org/wiki/Muller-Lyer_illusion">wikipedia</a></td>
<td><strong>Note:</strong> the task requires a quick response, thus attention abilities can be hard to measure in this task.</td>
</tr>
</tbody>
</table>

| **Oddball Task**                              | Respond to a stimulus dimension overshadowed by irrelevant dimension.       |
| Version of Huettel's implementation           | **Key Skills used:** Inhibition, Selective Attention, Visual Processing, Reaction Time. |
|                                                | **Note:** Inhibition skills require the individual to ignore the location and instead focus on the shape differences. |

| **Simon Task**                                | Respond to a stimulus dimension, overshadowed by spatial location.          |
| Simon's S-R compatibility test                | **Key Skills used:** Color Processing, Inhibition, Visual Processing, Selective Attention, Reaction Time. |
|                                                | **Note:** those who are color blind will have difficulty in completing this task. Individual's inhibition abilities can be measured (via correct responses) to see how well they can focus on the point of the task (color) and not be distracted by its location. |
Chapter 7. The PEBL Psychological Test Battery

### Switcher Task

**Description:**
Respond to a matched and changing stimulus dimension.

**Key Skills used:** Visual Processing, Selective Attention.

**Note:** reaction time is measured in the data section, along with trail type. Thus, times associated with color, shape and letter can be measured to see which is easiest and most challenging for the individual.

### Norms and Other Uses

Many of the original versions of the tasks we implement here have been normed on a large population. Such norms are available in published articles. Because these implementations are not identical (many of them use slightly different stimuli, response methods, timing, etc.) one must be careful when applying the results to the normed data. If you use PEBL or the PEBL Psychological Test Battery, please reference us! If you are interested in helping develop norms for PEBL tests, have access to subject populations and testing facilities, join the pebl-norms@lists.sourceforge.net mailing list and tell us what norms you are most interested in.

### Support and Contact info

If you have any general questions about PEBL or the PEBL Psychological Test Battery, you can contact us at: pebl-list@lists.sourceforge.net. Email support is available free-of-charge. You can sign up for this email list or browse the archives here. More information about the main author is available here. Enquire on the list if you are interested in paying someone to write new experiments or modify existing ones for your needs.

### Obtaining the Battery

The PEBL Test Battery is installed with the main PEBL installation. The first time you run PEBL, it will be copied into a folder in your Documents directory called pebl-exp.0.14 (or similar depending on the version of PEBL you are running). On Linux, running `pebl -install` will copy the battery directory there. The PEBL launcher will start in that directory, and let you explore and navigate the different tests in the battery.

[http://pebl.sourceforge.net/battery.html](http://pebl.sourceforge.net/battery.html)
Chapter 8

Detailed Function and Keyword Reference

8.1 Symbols

Name/Symbol: +

Description: Adds two expressions together. Also, concatenates strings together.

Usage:

\[ <\text{num1}> + <\text{num2}> \]
\[ <\text{string1}> + <\text{string2}> \]
\[ <\text{string1}> + <\text{num1}> \]

Using other types of variables will cause errors.

Example:

\[ 33 + 322 \rightarrow 355 \]
\[ "Hello" + " " + "World" \rightarrow "Hello World" \]
\[ "Hello" + 33 + 322.5 \rightarrow "Hello355.5" \]
\[ 33 + 322.5 + "Hello" \rightarrow "33322.5Hello" \]

See Also: -, ToString()
Chapter 8. Detailed Function and Keyword Reference

See Also:

Name/Symbol: /
Description: Divides one expression by another
Usage: <expression> / <expression>
Example: 333 / 10 # == 33.3
See Also:

Name/Symbol: *
Description: Multiplies two expressions together
Usage: <expression> * <expression>
Example: 32 * 2 # == 64
See Also:

Name/Symbol: ^
Description: Raises one expression to the power of another expression
Usage: <expression> ^ <expression>
Example: 25 ^ 2 # == 625
See Also: Exp, NthRoot

Name/Symbol: ;
Description: Finishes a statement, can start new statement on the same line (not needed at end of line)
Usage:
Example:
See Also:

Name/Symbol: #
**Chapter 8. Detailed Function and Keyword Reference**

- **Description:** Comment indicator; anything until the next CR following this character is ignored
  
- **Usage:**

- **Example:**

- **See Also:**

---

**Name/Symbol:** `<-`

**Description:** The assignment operator. Assigns a value to a variable

N.B.: This two-character sequence takes the place of the `=` operator found in many programming languages.

**Usage:**

**Example:**

**See Also:**

---

**Name/Symbol:** `( )`

**Description:** Groups mathematical operations

**Usage:**

( expression )

**Example:**

(3 + 22) * 4 # == 100

**See Also:**

---

**Name/Symbol:** `{ }`

**Description:** Groups a series of statements

**Usage:**

{ statement1
  statement2
  statement3
}

**Example:**

**See Also:**

---

**Name/Symbol:** `[ ]`
Chapter 8. Detailed Function and Keyword Reference

Description: Creates a list. Closing ] must be on same line as last element of list, even for nested lists.
Usage: [<item1>, <item2>, ....]
Example: [] #Creates an empty list
        [1,2,3] #Simple list
        [[3,3,3],[2,2],0] #creates a nested list structure
See Also: List()

Name/Symbol: <
Description: Less than. Used to compare two numeric quantities.
Usage: 3 < 5
        3 < value
Example: if(j < 33)
        {
            Print ("j is less than 33.")
        }
See Also: >, >=, <=, ==, !=, <>

Name/Symbol: >
Description: Greater than. Used to compare two numeric quantities.
Usage: 5 > 3
        5 > value
Example: if(j > 55)
        {
            Print ("j is greater than 55.")
        }
See Also: <, >=, <=, ==, !=, <>

Name/Symbol: <=
Description: Less than or equal to.
Usage: 3<=5
        3<=value
Chapter 8. Detailed Function and Keyword Reference

Example:
```java
if(j <= 33)
{
    Print ("j is less than or equal to 33.")
}
```

See Also: `<`, `>`, `>=`, `==`, `~=``, `!=`, `<>

Name/Symbol: `>=`
Description: Greater than or equal to.
Usage:
```
5>=3
5>=value
```
Example:
```
if(j >= 55)
{
    Print ("j is greater than or equal to 55.")
}
```

See Also: `<`, `>`, `<=`, `==`, `~=``, `!=`, `<>

Name/Symbol: `==`
Description: Equal to.
Usage:
```
4 == 4
```
Example:
```
2 + 2 == 4
```

See Also: `<`, `>`, `>=`, `<=`, `~=``, `!=`, `<>

Name/Symbol: `<>`, `!=`, `~=`
Description: Not equal to.
Usage:

Example:

See Also: `<`, `>`, `>=`, `<=`, `==

95
Name/Symbol: Abs()
Description: Returns the absolute value of the number.
Usage: Abs(<num>)
Example: Abs(-300) # ==300
Abs(23) # ==23
See Also: Round(), Floor(), AbsFloor(), Sign(), Ceiling()

Name/Symbol: AbsFloor()
Description: Rounds <num> toward 0 to an integer.
Usage: AbsFloor(<num>)
Example: AbsFloor(-332.7) # == -332
AbsFloor(32.88) # == 32
See Also: Round(), Floor(), Abs(), Sign(), Ceiling()

Name/Symbol: ACos()
Description: Inverse cosine of <num>, in degrees.
Usage: ACos(<num>)

Name/Symbol: AddObject()
**Chapter 8. Detailed Function and Keyword Reference**

**Description:** Adds a widget to a parent window, at the top of the object stack. Once added, the object will be drawn onto the parent last, meaning it will be on top of anything previously added.

In general, objects can be added to other objects as well as windows. For example, you can add drawing objects (circles, etc.) to an image to annotate the image and maintain its proper x,y coordinates.

Also, if you 're-add' an object that is already on a widget, it will get automatically removed from the window first. This is an easy way to reorder elements on a screen.

```
AddObject(<obj>, <window>)
AddObject(<obj>, <canvas>)
AddObject(<obj>, <widget>)
```

**Example:**

```
define Start(p)
{
    win <- MakeWindow()
    img <- MakeImage("pebl.png")
    circ <- Circle(20,20,10,MakeColor("red"),1)
    AddObject(circ,img)
    AddObject(img,win)
    Move(img,100,100)
    Draw()
    WaitForAnyKeyPress()
}
```

**See Also:** RemoveObject()

---

**Name/Symbol:** `and`

**Description:** Logical and operator.

**Usage:** `<expression> and <expression>`

**Example:**

**See Also:** `or`, `not`

---

**Name/Symbol:** `Append`
Chapter 8. Detailed Function and Keyword Reference

Description: Appends an item to a list. Useful for constructing lists in conjunction with the loop statement.

Note: `Append()` is useful, but inefficient for large data structures, because it requires making a copy of the entire data list and then overwriting it, if you use `list <- Append(list, item)`. The overhead will be hardly noticeable unless you are building lists hundreds of elements long. In that case you should either create the list upfront and use `SetElement`, or you `PushOnEnd` to modify the list directly.

Usage: `Append(<list>, <item>)`

Example:
```
list <- Sequence(1,5,1)
double <- []
loop(i, list)
{
    double <- Append(double, [i,i])
}
Print(double)
# Produces [[1,1],[2,2],[3,3],[4,4],[5,5]]
```

See Also: `SetElement()`, `List()`, `[]`, `Merge()`, `PushOnEnd`

Name/Symbol: `AppendFile`

Description: Appends onto the end of `<file1>` the contents of `<file2>`. Useful for compiling pooled data at the end of an experiment.

Usage: `AppendFile(<file1>, <file2>)`

Example:
```
The following open ten consecutive files, writes 50 random numbers to each, then appends each to a master file:

```
loop(j, Sequence(1,10,1))
{
    file <- FileOpenWrite(j".txt")
    loop(i,Sequence(1,50,1))
    {
        FilePrint(file,j"","+i","+Random()")
    }
    AppendFile("master.txt",j".txt")
}
```

See Also: `FileOpenWrite()`
**Chapter 8. Detailed Function and Keyword Reference**

Name/Symbol: ASin()

Description: Inverse Sine of \(<\text{num}>\), in degrees.

Usage: \(\text{ASin}(\text{<num>})\)

Example:

See Also: Cos(), Sin(), Tan(), ATan(), ACos(), ATan()

Name/Symbol: ATan

Description: Inverse Tan of \(<\text{num}>\), in degrees.

Usage:

Example:

See Also: Cos(), Sin(), Tan(), ATan(), ACos(), ATan()
Name/Symbol: Bezier

Description: Creates a smoothed line through the points specified by `<xpoints>`, `<ypoints>`. The lists `<xpoints>` and `<ypoints>` are adjusted by `<x>` and `<y>`, so they should be relative to 0, not the location you want the points to be at.

Like other drawn objects, the bezier must then be added to the window to appear. `<steps>` denotes how smooth the approximation will be.

Usage: 
```
Beizer(<x>,<y>,<xpoints>,<ypoints>,
    <steps>,<color>)
```

Example: 
```
win <- MakeWindow()
#This makes a T
xpoints <- [-10,10,10,20,20,-20,-20,-10]
ypoints <- [-20,-20,40,40,50,50,40,40]
p1 <- Beizer(100,100,xpoints, ypoints,
             5, MakeColor("black"))
AddObject(p1,win)
Draw()
```

See Also: BlockE(), Polygon(), MakeStarPoints(), MakeNGonPoints()

Name/Symbol: BlockE

Description: Creates a polygon in the shape of a block E, pointing in one of four directions. Arguments include position in window.

- `<x>` and `<y>` is the position of the center
- `<h>` and `<w>` or the size of the E in pixels
- `<thickness>` thickness of the E
- `<direction>` specifies which way the E points: 1=right, 2=down, 3=left, 4=up.
- `<color>` is a color object (not just the name)

Like other drawn objects, the Block E must then be added to the window to appear.

Usage: 
```
BlockE(x,y,h,w,thickness,direction,color)
```
Chapter 8. Detailed Function and Keyword Reference

Example:

```plaintext
win <- MakeWindow()
e1 <- BlockE(100,100,40,80,10,1,MakeColor("black"))
AddObject(e1,win)
Draw()
```

See Also: Plus(), Polygon(), MakeStarPoints(), MakeNGonPoints()

Name/Symbol: break

Description: Breaks out of a loop immediately.

Usage: break

Example:

```plaintext
loop(i , [1,3,5,9,2,7])
{
    Print(i)
    if(i == 3)
    {
        break
    }
}
```

See Also: loop, return
8.4 C

Name/Symbol: Ceiling()
Description: Rounds <num> up to the next integer.
Usage: Ceiling(<num>)
Example: Ceiling(33.23)  # == 34
         Ceiling(-33.02) # == -33
See Also: Round(), Floor(), AbsFloor(), Ceiling()

Name/Symbol: ChooseN()
Description: Samples <number> items from list, returning a list in the original order. Items are sampled without replacement, so once an item is chosen it will not be chosen again. If <number> is larger than the length of the list, the entire list is returned in order. It differs from SampleN in that ChooseN returns items in the order they appeared in the original list, but SampleN is shuffled.
Usage: ChooseN(<list>, <n>)
Example:
# Returns 5 numbers
ChooseN([1,1,1,2,2], 5)

# Returns 3 numbers from 1 and 7:
ChooseN([1,2,3,4,5,6,7], 3)
See Also: SampleN(), SampleNWithReplacement(), Subset()

Name/Symbol: Circle()
Description: Creates a circle for graphing at x,y with radius r. Circles must be added to a parent widget before it can be drawn; it may be added to widgets other than a base window. The properties of circles may be changed by accessing their properties directly, including the FILLED property which makes the object an outline versus a filled shape.
Usage: Circle(<x>, <y>, <r>,<color>)
Example:
\[
c \leftarrow \text{Circle}(30, 30, 20, \text{MakeColor}(\text{green}))
\]
\[
\text{AddObject}(c, \text{win})
\]
\[
\text{Draw}()
\]

See Also: \text{Square()}, \text{Ellipse()}, \text{Rectangle()}, \text{Line}()

Name/Symbol: \text{CheckForNetworkConnection()}

Description: Checks to see if there is an incoming TCP/IP connection on a network that is opened using \text{OpenNetworkListener}. This is an alternative to the \text{WaitForNetworkConnection} function that allows more flexibility (and allows updating the during waiting for the connection).

Usage:
\[
\text{net} \leftarrow \text{CheckForNetworkConnection}(\text{network})
\]

Example:
\[
\text{network} \leftarrow \text{OpenNetworkListener}(4444)
\]
\[
\text{time} \leftarrow \text{GetTime}()
\]
\[
\text{while(not connected and (GetTime()} < \text{time} + 5000))
\]
\[
\begin{align*}
\{ & \text{connected} \leftarrow \text{CheckForNetworkConnection}(\text{network}) \\
\}
\end{align*}
\]

See Also: \text{OpenNetworkListener()}, \text{Getdata()}, \text{WaitForNetworkConnection()}, \text{CloseNetwork()}

Name/Symbol: \text{ClearEventLoop()}

Description: Clears the event loop. This function is currently experimental, and its usage may change in future versions of PEBL.

Usage: \text{USAGE CURRENTLY UNDOCUMENTED}

Example:

See Also: \text{RegisterEvent()}, \text{StartEventLoop()}

Name/Symbol: \text{ClickCheckbox()}
Chapter 8. Detailed Function and Keyword Reference

Description: This 'clicks' a checkbox, changing its status (both the visual display and its .status property). Its state can also be set using the SetCheckBox() function. The text "ClickCheckBox" is by default bound to the .clickon property of any checkbox, enabling you to handle on a number of graphical object the same (see callfunction example). The [x,y] coordinates are ignored, and so anything can be fed to them, but the standard approach is to use give gClick, which is a global bound to the last click coordinates when WaitForClickOnTarget is used.

Usage:    ClickCheckBox(obj, [x,y])

Example: The following creates a button, waits for you to click on it, and animates a button press

    ok <- MakeCheckbox("OK?",400,250,gWin,150)
    resp <- WaitForClickOnTarget([ok],[1])
    ClickCheckBox(done,gClick)
    Draw()

You can handle a bunch of objects together using an idiom like this:

    ok <- MakeCheckbox("OK?",400,250,gWin,150)
    ok2 <- MakeCheckbox("Otherwise?",400,280,gWin,150)

    checks  <- [ok,ok2]
    resp <- WaitForClickOnTarget(checks,[1,2])
    check <- Nth(checks,resp)
    CallFunction(check.clickon,[check,gClick])
    Draw()

Examples of its use can be found in demo ui.pbl

See Also:    MakeCheckBox(), SetCheckBox()

Name/Symbol: Clickon()

Description: Calls the function named by the .clickon property of a custom object. Useful for handling click events of a bunch of different objects. This is essentially the same as CallFunction(obj.clickon, [obj,gClick]).

Usage:    Clickon(obj,[x,y])
Example:

```plaintext
## This overrides buttons placement at the center:
done <- MakeButton("QUIT",400,250,gWin,150)
WaitForClickOnTarget([done],[1])
ClickOn(done,gClick)
```

See Also: `Inside()`, `ClickCheckbox`, `MoveObject`, `DrawObject`

Name/Symbol: `ClickOnMenu()`

Description: Handles clicking on a menu item. It will call the `click` property of that item, and then hide the menu.

Usage:

```plaintext
ClickOnMenu(obj,[x,y])
```

This function is typically not used directly, but rather it is called via `MakeMenu`. However, it can be used as a quick-and-dirty button.

Example:

This creates a menu and awaits clicking on. More complete examples are available in ui.pbl. It requires that `MyMessage` is created somewhere:

```plaintext
menu1 <- MakeMenuItem("File",0,0,gWin,14,10,"MYMESSAGE")

menu2<- MakeMenu("Edit",70,0,gWin,14,10, "MYMESSAGE")

menus <- [menu1,menu2]
opt <- WaitForClickOnTarget(menu,[1,2])
ClickOnMenu(Nth(menus,opt),gClick)
```

See Also: `MakeMenu()`, `OpenSubMenus()`, `MakeMenuItem`

Name/Symbol: `ClickOnScrollbox()`

Description: Handles a click event on the a `ScrollBox`. This should be called after one checks (e.g., via `InsideTB`) whether the scrollbox was actually clicked on. It will handle scrolling, moving via the thumb, up/down arrows, and reselection. It is also used to interact with `ScrollingTextBox` objects. This function name is bound to the `click` property of scrollboxes, so it can be called using `CallFunction` (see example below).

105
Chapter 8. Detailed Function and Keyword Reference

Usage: \texttt{ClickOnScrollbox(sb,[x,y])}

Here, \texttt{sb} is the scrollbox object. \texttt{[x,y]} is a list of xy coordinates, which can also be the global variable \texttt{gClick}

Example: See \texttt{ui.pbl} in the demo directory for examples of the use of a scrolling text box. A brief example follows:

\begin{verbatim}
sb <- MakeScrollBox(Sequence(1,50,1),"The numbers",40,40,gWin,12,150,500,3)
Draw()

resp <- WaitForClickOnTarget([sb],[1])
ClickOnScrollbox(sb,gClick)

#Alternately: CallFunction(sb.clickon,[sb,gClick])

##change the selected items
sb.list <- Sequence(sb.selected,sb.selected+50,1)
UpdateScrollbox(sb)
DrawScrollbox(sb)
Draw()
\end{verbatim}

See Also: \texttt{MakeScrollingTextBox \ MakeScrollBox \ UpdateScrollBox \ DrawScrollBox}

Name/Symbol: \texttt{CloseNetworkConnection()}

Description: Closes network connection

Usage: \texttt{CloseNetwork(<network>)}

Example: \begin{verbatim}
net <- WaitForNetworkConnection("localhost",1234)
SendData(net,"Watson, come here. I need you.")
CloseNetworkConnection(net)
\end{verbatim}

Also see \texttt{nim.pbl} for example of two-way network connection.

See Also: \texttt{ConnectToIP, ConnectToHost, WaitForNetworkConnection, GetData, SendData, ConvertIPString}

Name/Symbol: \texttt{ConnectToHost()}

106
**Chapter 8. Detailed Function and Keyword Reference**

**Description:** Connects to a host computer waiting for a connection on `<port>`, returning a network object that can be used to communicate. Host is a text hostname, like "myname.indiana.edu", or use "localhost" to specify your current computer.

**Usage:**

ConnectToHost(<hostname>,<port>)

**Example:** See nim.pbl for example of two-way network connection.

```r
net <- ConnectToHost("localhost",1234)
dat <- GetData(net,20)
Print(dat)
CloseNetworkConnection(net)
```

**See Also:** ConnectToIP, GetData, WaitForNetworkConnection, SendData, ConvertIPString, CloseNetworkConnection

---

**Name/Symbol:** ConnectToIP()

**Description:** Connects to a host computer waiting for a connection on `<port>`, returning a network object that can be used to communicate. `<ip>` is a numeric ip address, which must be created with the ConvertIPString(ip) function.

**Usage:**

ConnectToIP(<ip>,<port>)

**Example:** See nim.pbl for example of two-way network connection.

```r
ip <- ConvertIPString("192.168.0.1")
net <- ConnectToHost(ip,1234)
dat <- GetData(net,20)
Print(dat)
CloseNetworkConnection(net)
```

**See Also:** ConnectToHost, GetData, WaitForNetworkConnection, SendData, ConvertIPString, CloseNetworkConnection

---

**Name/Symbol:** ConvertIPString()

**Description:** Converts an IP address specified as a string into an integer that can be used by ConnectToIP.

**Usage:**

ConvertIPString(<ip-as-string>)

**Example:** See nim.pbl for example of two-way network connection.
Chapter 8. Detailed Function and Keyword Reference

ip <- ConvertIPString("192.168.0.1")
net <- ConnectToHost(ip,1234)
dat <- GetData(net,20)
Print(dat)
CloseNetworkConnection(net)

See Also: ConnectToHost, ConnectToIP, GetData,
WaitForNetworkConnection, SendData,
CloseNetworkConnection

Name/Symbol: ConvexHull()

Description: Computes the convex hull of a set of [x,y] points. It returns a
set of points that forms the convex hull, with the first and last
point identical. A convex hull is the set of outermost points,
such that a polygon connecting just those points will encompass
all other points, and such that no angle is acute. It is used in
MakeAttneave.

Usage: ConvexHull(<list-of-x-y-points>)

Example: pts <- [[0.579081, 0.0327737],
                  [0.0536094, 0.378258],
                  [0.239628, 0.187751],
                  [0.940625, 0.26526],
                  [0.508748, 0.840846],
                  [0.352604, 0.200193],
                  [0.38684, 0.212413],
                  [0.00114761, 0.768165],
                  [0.432963, 0.629412]]
Print(ConvexHull(pts))

output:
[[0.940625, 0.26526],
 [0.508748, 0.840846],
 [0.00114761, 0.768165],
 [0.0536094, 0.378258],
 [0.239628, 0.187751],
 [0.579081, 0.0327737],
 [0.940625, 0.26526]]

See Also: MakeAttneave,
Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: Cos()
Description: Cosine of <deg> degrees.
Usage:
Example:
Cos(33.5)
Cos(-32)
See Also: Sin(), Tan(), ATan(), ACos(), ATan()

Name/Symbol: CountDown()
Description: Displays a 3-2-1 countdown on the screen in with 500 ms ISI. CountDown temporarily hides whatever is on the screen. It is useful in orienting participants to the first trial of a task.
Usage:
Example:
win <- MakeWindow()
MessageBox("Press any key to begin", win)
CountDown(win)
Trial()
See Also: MessageBox

Name/Symbol: CR()
Description: Produces <number> linefeeds which can be added to a string and printed or saved to a file. CR is an abbreviation for “Carriage Return”.
Usage:
Example:
Print("Number: " Tab(1) + number + CR(2))
Print("We needed space before this line.")
See Also: Format(), Tab()

Name/Symbol: CrossFactorWithoutDuplicates()
Description: This function takes a single list, and returns a list of all pairs, excluding the pairs that have two of the same item. To achieve the same effect but include the duplicates, use: DesignFullCounterBalance(x,x).
Chapter 8. Detailed Function and Keyword Reference

Usage: `CrossFactorWithoutDuplicates(<list>)`

Example: `CrossFactorWithoutDuplicates([a,b,c])`
# == [[a,b],[a,c],[b,a],[b,c],[c,a],[c,b]]

See Also: `DesignFullCounterBalance()`, `Repeat()`, `DesignBalancedSampling()`, `DesignGrecoLatinSquare()`, `DesignLatinSquare()`, `RepeatList()`, `LatinSquare()`, `Shuffle()`

Name/Symbol: `CumNormInv()`

Description: This function takes a probability and returns the corresponding z-score for the cumulative standard normal distribution. It uses an accurate numerical approximation from:
http://home.online.no/pjacklam/notes/invnorm

Usage: `CumNormInv(<p>)`

Example:
```
Print(CumNormInv(0))  #= NA
Print(CumNormInv(.01)) #= -2.32634
Print(CumNormInv(.5))  #= 0
Print(CumNormInv(.9))  #= 1.28
Print(CumNormInv(1))   #= NA
```

See Also: `NormalDensity()`, `RandomNormal()`
### 8.5 D

**Name/Symbol:** `define`  
**Description:** Defines a user-specified function.  
**Usage:**  
```plaintext
define functionname (parameters)  
{  
    statement1  
    statement2  
    statement3  
    #Return statement is optional:  
    return <value>  
}
```

**Example:**  
See above.

**See Also:**

**Name/Symbol:** `DegToRad()`  
**Description:** Converts degrees to radians.  
**Usage:**  
```plaintext
DegToRad(<deg>)
```

**Example:**  
```plaintext
DegToRad(180) # == 3.14159...
```

**See Also:**  
`Cos()`, `Sin()`, `Tan()`, `ATan()`, `ACos()`, `ATan()`

**Name/Symbol:** `DesignBalancedSampling()`  
**Description:** Samples elements “roughly” equally. This function returns a list of repeated samples from `<treatment_list>`, such that each element in `<treatment_list>` appears approximately equally. Each element from `<treatment_list>` is sampled once without replacement before all elements are returned to the mix and sampling is repeated. If there are no repeated items in `<list>`, there will be no consecutive repeats in the output. The last repeat-sampling will be truncated so that a `<length>`-size list is returned. If you don’t want the repeated epochs this function provides, `Shuffle()` the results.  
**Usage:**  
```plaintext
DesignBalancedSampling(<list>, <length>)
```
Example:  
DesignBalancedSampling([1,2,3,4,5],12) 
    ## e.g., produces something like:  
    ##    [5,3,1,4,2, 3,1,5,2,4, 3,1 ]

See Also:  
CrossFactorWithoutDuplicates(), Shuffle(), DesignFullCounterBalance(), 
DesignGrecoLatinSquare(), DesignLatinSquare(), Repeat(), RepeatList(), 
LatinSquare()

Name/Symbol:  
DesignFullCounterbalance()

Description:  
This takes two lists as parameters, and returns a nested list of 
lists that includes the full counterbalancing of both parameter 
lists. Use cautiously; this gets very large.

Usage:  
DesignFullCounterbalance(<lista>, <listb>)

Example:  
a <- [1,2,3]  
b <- [9,8,7]  
DesignFullCounterbalance(a,b)  
# == [[1,9], [1,8], [1,7],  
#      [2,9], [2,8], [2,7],  
#      [3,9], [3,8], [3,7]]

See Also:  
CrossFactorWithoutDuplicates(), LatinSquare(), Shuffle(), DesignBalancedSampling(), 
DesignGrecoLatinSquare(), DesignLatinSquare(), 
Repeat(), RepeatList(),

Name/Symbol:  
DesignGrecoLatinSquare()

Description:  
This will return a list of lists formed by rotating through each 
element of the <treatment_list>, making a list containing all 
element of the list, according to a greco-latin square. All lists 
must be of the same length.

Usage:  
DesignGrecoLatinSquare(<factor_list>,  
    <treatment_list>, <treatment_list>)

Example:  
x <- ["a","b","c"]  
y <- ["p","q","r"]  
z <- ["x","y","z"]  
Print(DesignGrecoLatinSquare(x,y,z))  
# produces:  
#   [[[a, p, x], [b, q, y], [c, r, z]],  
#    [[a, q, z], [b, r, x], [c, p, y]],  
#    [[a, r, y], [b, p, z], [c, q, x]]]
Chapter 8. Detailed Function and Keyword Reference

See Also: CrossFactorWithoutDuplicates(), LatinSquare(), DesignFullCounterBalance(), DesignBalancedSampling(), DesignLatinSquare(), Repeat(), RepeatList(), Shuffle()

Name/Symbol: DesignLatinSquare()

Description: This returns return a list of lists formed by rotating through each element of <treatment_list>, making a list containing all element of the list. Has no side effect on input lists.

Usage: DesignLatinSquare(<treatment1_list>, <treatment2_list>)

Example: order <- [1,2,3]
treatment <- ["A","B","C"]
design <- DesignLatinSquare(order,treatment)
# produces: [[[1, A], [2, B], [3, C]],
# [1, B], [2, C], [3, A]],
# [1, C], [2, A], [3, B]]

See Also: CrossFactorWithoutDuplicates(), DesignFullCounterBalance(), DesignBalancedSampling(), DesignGrecoLatinSquare(), Repeat(), LatinSquare(), RepeatList(), Shuffle(), Rotate()

Name/Symbol: Dist()

Description: Returns Euclidean distance between two points. Each point should be \([x,y]\), and any additional items in the list are ignored.

Usage: Dist(<xylist1>, <xylist2>)

Example: p1 <- [0,0]
p2 <- [3,4]
d <- Dist(p1,p2) #d is 5

See Also:

Name/Symbol: Div()

Description: Returns round\((\text{num}/\text{mod})\)

Usage: Div(<num>, <mod>)

Example:
Chapter 8. Detailed Function and Keyword Reference

See Also: Mod()

Name/Symbol: Draw()
Description: Redraws the screen or a specific widget.
Usage: Draw()
Example:
See Also: DrawFor(), Show(), Hide()

Name/Symbol: DrawFor()
Description: Draws a screen or widget, returning after <cycles> refreshes. This function currently does not work as intended in the SDL implementation, because of a lack of control over the refresh blank. It may work in the future.
Usage: DrawFor( <object>, <cycles>)
Example:
See Also: Draw(), Show(), Hide()

Name/Symbol: DrawObject()
Description: Calls the function named by the .draw property of a custom object. Useful for handling drawing of a bunch of different objects. This is essentially the same as CallFunction(obj.draw, [obj]), but falls back to a normal Draw() command so it handles built-in objects as well.
Usage: DrawObject(obj)
Example:

##This overrides buttons placement at the center:
done <- MakeButton("QUIT",400,250,gWin,150)
WaitForClickOnTarget([done],[1])
Clickon(done,gClick)
DrawObject(done)

See Also: Inside(), ClickOnCheckbox MoveObject, Draw
**Chapter 8. Detailed Function and Keyword Reference**

Name/Symbol: **DrawPullDown()**

**Description:** This handles layout/drawing of a pulldown box. This does not actually call Draw() on the window, and so an additional draw command is needed before the output is displayed. The main use case for this function is if you need to manually change the selected object (by changing .selected). This will redraw the pulldown with the new selection.

**Usage:**

```r
DrawPullDown(object)
```

**Example:**

```r
options <- MakePulldownButton(["A","B","C"],400,250,gWin,14,100,1)
Draw()
WaitForAnyKeyPress()
options.selected <- 2
DrawPullDown(options)
Draw()
WaitForAnyKeyPress()
```

See Also: **MakePullDown(), Pulldown(), UpdatePulldown**

---

Name/Symbol: **DrawScrollbox()**

**Description:** Redraws a ScrollBox. This is called by various internal functions, but should be used to handle redrawing if UpdateScrollbox is used. When things like the scrollbar, offset, and selected item change, this can be called directly. If the actual list is changed, UpdateScrollBox should be called first. Note that the redrawn scrollbox won't be changed on the screen until a Draw() command is issued.

**Usage:**

```r
DrawScrollBox(sb)
```

Here, sb is the scrollbox object.

**Example:**

See ui.pbl in the demo directory for examples of the use of a scrolling text box. A brief example follows:

```r
sb <- MakeScrollBox(Sequence(1,50,1),"The numbers",40,40,gWin,12,150,500,3)
Draw()

resp <- WaitForClickOntarget([sb],[1])
CallFunction(sb.clickon,[sb,gClick])
```

115
#Alternately: ClickOnScrollbox(sb,gClick)

##change the selected items
sb.list <- Sequence(sb.selected,sb.selected+50,1)
UpdateScrollbox(sb)
DrawScrollbox(sb)
Draw()

See Also: MakeScrollingTextBox MakeScrollBox UpdateScrollBox ClickOnScrollBox
Chapter 8. Detailed Function and Keyword Reference

8.6 E

Name/Symbol: EasyLabel()

Description: Creates and adds to the window location a label at specified location. Uses standard vera font with grey background. (May in the future get background color from window). Easy-to-use replacement for the MakeFont, MakeLabel, AddObject, Move, steps you typically have to go through.

Usage: EasyLabel( <text>, <x>, <y>, <win>, <fontsize>)

Example:

```
win <- MakeWindow()
lab <- EasyLabel("What?", 200, 100, win, 12)
Draw()
```

See Also: EasyTextBox(), MakeLabel()

Name/Symbol: EasyTextBox()

Description: Creates and adds to the window location a textbox at specified location. Uses standard vera font with white background. Easy-to-use replacement for the MakeFont, MakeTextBox, AddObject, Move, steps.

Usage: EasyTextBox( <text>, <x>, <y>, <win>, <fontsize>, <width>, <height>)

Example:

```
win <- MakeWindow()
entry <- EasyTextBox("1 2 3 4 5", 200, 100, win, 12, 200, 50)
Draw()
```

See Also: EasyLabel(), MakeTextBox()

Name/Symbol: Ellipse()

Description: Creates a ellipse for graphing at x,y with radii rx and ry. Ellipses are only currently definable oriented in horizontal/vertical directions. Ellipses must be added to a parent widget before it can be drawn; it may be added to widgets other than a base window. The properties of ellipses may be changed by accessing their properties directly, including the FILLED property which makes the object an outline versus a filled shape.
Chapter 8. Detailed Function and Keyword Reference

Usage: `Ellipse(<x>, <y>, <rx>, <ry>, <color>)`

Example:
```plaintext
e <- Ellipse(30, 30, 20, 10, MakeColor(green))
AddObject(e, win)
Draw()
```

See Also: `Square()`, `Circle()`, `Rectangle()`, `Line()`

---

Name/Symbol: `EndOfFile()`
Description: Returns true if at the end of a file.
Usage: `EndOfFile(<filestream>)`
Example:
```plaintext
while(not EndOfFile(fstream))
{
    Print(FileReadLine(fstream))
}
```

See Also:

---

Name/Symbol: `EndOfLine()`
Description: Returns true if at end of line.
Usage: `EndOfLine(<filestream>)`
Example:

See Also:

---

Name/Symbol: `Enquote()`
Description: Surrounds the argument with quotes.
Usage: `Enquote("one two three")`
Example:
```plaintext
##use to add quoted text to instructions.
instructions <- 'Respond whenever you see an "' +
                  Enquote('X')
```
```plaintext
##Use it for saving data that may have spaces:
resp <- GetInput(tb, "<enter>")
FilePrint(fileout, Enquote(resp))
```

118
See Also: gQuote

Name/Symbol: Exp()
Description: \( e \) to the power of \(<pow>\).
Usage: \( \text{Exp}(<pow>) \)
Example: \( \text{Exp}(0) \) # == 1  
          \( \text{Exp}(3) \) # == 20.0855
See Also: Log()
8.7 F

Name/Symbol: FileClose()

Description: Closes a filestream variable. Be sure to pass the variable name, not the filename.

Usage: FileClose(<filestream>)

Example: x <- FileOpenRead("file.txt")
# Do relevant stuff here.
FileClose(x)

See Also: FileOpenAppend(), FileOpenRead(), FileOpenWrite()

Name/Symbol: FileExists()

Description: Checks whether a file exists. Returns 1 if it exists, 0 otherwise.

Usage: FileExists(<path>)

Example: filename <- "data-"+gSubNum+".csv"
exists <- FileExists(filename)
if(exists)
    {  
        MessageBox("Subject file already exists. Please try a new one.",gWin
        SignalFatalError("filename already used")
    }

See Also: GetDirectoryListing(), FileExists(), IsDirectory(), MakeDirectory()

Name/Symbol: FileOpenAppend()

Description: Opens a filename, returning a stream that can be used for writing information. Appends if the file already exists.

Usage: FileOpenAppend(<filename>)

Example:

See Also: FileClose(), FileOpenRead(), FileOpenWrite(), FileOpenOverWrite()
Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: FileOpenOverWrite()

Description: Opens a filename, returning a stream that can be used for writing information. Overwrites if file already exists. This function should not be used for opening data files; instead, use FileOpenWrite, which saves to a backup file if the specified file already exists.

Usage: FileOpenOverWrite(<filename>)

Example:

See Also: FileClose(), FileOpenAppend(), FileOpenRead(), FileOpenWrite()

Name/Symbol: FileOpenRead()

Description: Opens a filename, returning a stream to be used for reading information.

Usage: FileOpenRead(<filename>)

Example:

See Also: FileClose(), FileOpenAppend(), FileOpenWrite(), FileOpenOverWrite()

Name/Symbol: FileOpenWrite()

Description: Opens a filename, returning a stream that can be used for writing information. If the specified filename exists, it won't overwrite that file. Instead, it will create a related filename, appending a -integer before the filename extension.

Usage: FileOpenWrite(<filename>)

Example: In the following example, test.txt gets created with the text “testing 1”, and then a second file test-1.txt gets created with the text “testing 2”.

```r
f1 <- FileOpenWrite("test.txt")
FilePrint(f1,"testing 1")
FileClose(f1)
f2 <- FileOpenWrite("test.txt")
FilePrint(f2,"testing 2")
FileClose(f2)
```
Chapter 8. Detailed Function and Keyword Reference

See Also: FileClose(), FileOpenAppend(), FileOpenRead(), FileOpenOverWrite()

Name/Symbol: FilePrint()
Description: Like Print, but to a file. Prints a string to a file, with a carriage return at the end. Returns a copy of the string it prints.
Usage: FilePrint(<filestream>, <value>)
Example: FilePrint(fstream, "Another Line.")
See Also: Print(), FilePrint_()

Name/Symbol: FilePrint_()
Description: Like Print_, but to a file. Prints a string to a file, without appending a newline character. Returns a copy of the string it prints.
Usage: FilePrint_(<filestream>, <value>)
Example: FilePrint_(fstream, "This line doesn't end.")
See Also: Print_(), FilePrint()

Name/Symbol: FilePrintList()
Description: Prints a list to a file, without the ’,’s or [] characters. Puts a carriage return at the end. Returns a string that was printed. If a list contains other lists, the printing will wrap multiple lines and the internal lists will be printed as normal. To avoid this, try FilePrintList(file, Flatten(list)).
Usage: FilePrintList(<filestream>, <list>)
Example:
FilePrintList(fstream, [1,2,3,4,5,5,5])
###
### Produces:
#1 2 3 4 5 5 5
FilePrintList(fstream,[[1,2],[3,4],[5,6]])
# Produces:
# [1,2]
#,[3,4]
Chapter 8. Detailed Function and Keyword Reference

```
#,[5,6]
FilePrintList(fstream, Flatten([[1,2],[3,4],[5,6]]))
#Produces:
# 1 2 3 4 5 6
```

See Also: Print(), Print_(), FilePrint(), FilePrint_(), PrintList(),

---

Name/Symbol: FileReadCharacter()

Description: Reads and returns a single character from a filestream.

Usage: FileReadCharacter(<filestream>)

Example:

See Also: FileReadList(), FileReadTable(), FileReadLine(), FileReadText(), FileReadWord().

---

Name/Symbol: FileReadLine()

Description: Reads and returns a line from a file; all characters up until the next newline or the end of the file.

Usage: FileReadLine(<filestream>)

Example:

See Also: FileReadCharacter(), FileReadList(), FileReadTable(), FileReadText(), FileReadWord().

---

Name/Symbol: FileReadList()

Description: Given a filename, will open it, read in all the items into a list (one item per line), and close the file afterward. Ignores blank lines or lines starting with #. Useful with a number of pre-defined data files stored in media/text/. See Section 4.18.4: Provided Media Files.

Usage: FileReadList(<filename>)

Example: FileReadList("data.txt")

123
See Also: FileReadCharacter(), FileReadTable(), FileReadLine(), FileReadText(), FileReadWord()

Name/Symbol: FileReadTable()
Description: Reads a table directly from a file. Data in file should be separated by spaces. Reads each line onto a sublist, with space-separated tokens as items in sublist. Ignores blank lines or lines beginning with #. Optionally, specify a token separator other than space.
Usage: FileReadTable(<filename>, <optional-separator>)
Example: a <- FileReadTable("data.txt")
See Also: FileReadCharacter(), FileReadList(), FileReadLine(), FileReadText(), FileReadWord()

Name/Symbol: FileReadText()
Description: Returns all of the text from a file, ignoring any lines beginning with #. Opens and closes the file transparently.
Usage: FileReadText(<filename>)
Example: instructions <- FileReadText("instructions.txt")
See Also: FileReadCharacter(), FileReadList(), FileReadTable(), FileReadLine(), FileReadText(), FileReadWord()

Name/Symbol: FileReadWord()
Description: Reads and returns a 'word' from a file; the next connected stream of characters not including a ' ' or a newline. Will not read newline characters.
Usage: FileReadWord(<filestream>)
Example:
See Also: FileReadLine(), FileReadTable(), FileReadList(), FileReadCharacter(), FileReadList(), FileReadTable(), FileReadLine(), FileReadText(), FileReadWord()
**Chapter 8. Detailed Function and Keyword Reference**

**Description:** Returns a subset of `<list>`, depending on whether the `<filter>` list is zero or nonzero. Both arguments must be lists of the same length.

**Usage:** `Filter(<list>,<filter>)`

**Example:**
```r
x <- c(1,2,3,3,2,2,1)
Print(Filter(x,c(1,1,1,0,0,0,0))) #==[1,2,3]
Print(Filter(x,Match(x,1))) #== [1,1]
```

**See Also:** `Match()`, `Subset()`, `Lookup()`

---

**Name/Symbol:** `FindInString()`

**Description:** Finds a token in a string, returning the position (starting at a particular position).

**Usage:** `FindInString(<basestring>,<searchstring>,<startingpos>)`

If the string is not found, the value 0 is returned.

**Example:**
```r
FindInString("about","bo",1) # == 2
FindInString("banana","na",1) # == 3
FindInString("banana","na",4) # == 5
```

**See Also:** `SplitString()`

---

**Name/Symbol:** `First()`

**Description:** Returns the first item of a list.

**Usage:** `First(<list>)`

**Example:**
```r
First([3,33,132]) # == 3
```

**See Also:** `Nth()`, `Last()`

---

**Name/Symbol:** `Flatten()`

**Description:** Flattens nested list `<list>` to a single flat list.

**Usage:** `Flatten(<list>)`

**Example:**
```r
Flatten([1,2,[3,4],[5,[6,7],8],[9]]) # == [1,2,3,4,5,6,7,8,9]
Flatten([1,2,[3,4],[5,[6,7],8],[9]]) # == [1,2,3,4,5,6,7,8,9]
```
Chapter 8. Detailed Function and Keyword Reference

See Also: FlattenN(), FoldList()

Name/Symbol: FlattenN()
Description: Flattens \( n \) levels of nested list \( \text{list} \).
Usage: Flatten(\text{list}, \text{n})
Example: Flatten([1,2,[3,4],[5,[6,7]],8],[9]],1)
# == [1,2,3,4,5,[6,7],8,9]
See Also: Flatten(), FoldList()

Name/Symbol: Floor()
Description: Rounds \( \text{num} \) down to the next integer.
Usage: Floor(\text{num})
Example: Floor(33.23) # == 33
Floor(3.999) # ==3
Floor(-32.23) # == -33
See Also: AbsFloor(), Round(), Ceiling()

Name/Symbol: FoldList()
Description: Folds a list into equal-length sublists.
Usage: FoldList(\text{list}, \text{size})
Example: FoldList([1,2,3,4,5,6,7,8],2)
# == [[1,2],[3,4],[5,6],[7,8]]
See Also: FlattenN(), Flatten()

Name/Symbol: Format()
Description: Formats the printing of values to ensure the proper spacing. It will either truncate or pad \( \text{value} \) with spaces so that it ends up exactly \( \text{length} \) characters long. Character padding is at the end.
Usage: Format(\text{value}, \text{length})
Example:

```r
x <- 33.23425225
y <- 23.3
Print("["+Format(x,5)+"]")
Print("["+Format(y,5)+"]")
## Output:
## [33.23 ]
## [23.3  ]
```

See Also: `CR()` `Tab()`
## Chapter 8. Detailed Function and Keyword Reference

### 8.8 G

<table>
<thead>
<tr>
<th>Name/Symbol:</th>
<th>Description:</th>
<th>Usage:</th>
<th>Example:</th>
</tr>
</thead>
</table>
| GetAngle()           | Gets an angle (in degrees) from (0,0) of an xy coordinate | GetAngle(<x>,<y>) | ```
#point sprite in the direction of a click
sprite <- LoadImage("car.png")
AddObject(sprite,gWin)
Move(sprite,300,300)
xy <- WaitForDownClick()
newangle <- GetAngle(First(xy)-300,Second(xy)-300)
sprite.rotation <- newangle
Draw()
``` |
|                      |                                                    |        | See Also: DegtoRad, RadToDeg                                                                 |

<table>
<thead>
<tr>
<th>Name/Symbol:</th>
<th>Description:</th>
<th>Usage:</th>
<th>Example:</th>
</tr>
</thead>
</table>
| GetAngle3()          | Gets an angle (in radians) of abc.                 | GetAngle3(<a>,<b>,<c>) | ```
a <- [0.579081, 0.0327737]
b <- [0.0536094, 0.378258]
c <- [0.239628, 0.187751]
Print(GetAngle3(a,b,c)) ## .2157
``` |
|                      |                                                    |        | See Also: DegtoRad, RadToDeg, GetAngle, ToRight                                               |

<table>
<thead>
<tr>
<th>Name/Symbol:</th>
<th>Description:</th>
<th>Usage:</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCurrentScreenResolution()</td>
<td>Returns an list of [width, height] specifying what the current computer screen resolution is. This is used within the pebl launcher in order to use the current resolution to run the experiment.</td>
<td>res &lt;- GetCurrentScreenResolution()</td>
<td></td>
</tr>
</tbody>
</table>
Example:

```nim
define Start(p)
{
    ## For testing, let's make the screen resolution a bit smaller than the
    ## current one so that it doesn't get hidden by the bottom task bar
    ##
    res <- GetCurrentScreenResolution()
    gVideoWidth <- First(res)-100
    gVideoHeight <- Second(res)-100
    gWin <- MakeWindow()
    MessageBox("Window slightly smaller than screen",gWin)
}
```

See Also: `GetVideoModes()`

Name/Symbol: `GetCursorPosition()`

Description: Returns an integer specifying where in a textbox the edit cursor is. The value indicates which character it is on.

Usage: `GetCursorPosition(<textbox>)`

Example:

See Also: `SetCursorPosition()`, `MakeTextBox()`, `SetText()`

Name/Symbol: `GetData()`

Description: Gets Data from network connection. Example of usage in `demo/nim.pbl`.

Usage: `val <- GetData(<network>,<size>)`

Example: On 'server':

```nim
net <- WaitForNetworkConnection("localhost",1234)
SendData(net,"Watson, come here. I need you.")
value <- GetData(net,10)
Print(value)
```

On Client:

```nim
net <- ConnectToHost("localhost",1234)
value <- GetData(net,20)
Print(value)
##should print out "Watson, come here. I need you."
```
See Also:  
ConnectToIP, ConnectToHost, WaitForNetworkConnection,  
SendData, ConvertIPString, CloseNetworkConnection

Name/Symbol: GetDirectoryListing()

Description:  Returns a list of files and directories in a particular directory/folder.

Usage:  
list <- GetDirectoryListing(<path>)

Example:  
files <- GetDirectoryListing("./")

See Also:  
GetDirectoryListing(), FileExists(), IsDirectory(),  
MakeDirectory()

Name/Symbol: GetEasyChoice()

Description:  Hides what is on the screen and presents a textbox with specified message, and a series of options to select from. Returns element from corresponding position of the <output> list.

Usage:  
GetEasyChoice(<message>,<list-of-choices>,  
<output>,<window>)

Example:  
The code snippet below produces the following screen:

```
gWin <- MakeWindow("white")
imp <- GetEasyChoice("What Year are you in school",  

["First-year","Sophomore",  
"Junior","Senior","Other"],  
[1,2,3,4,5],  gWin)
```

130
Chapter 8. Detailed Function and Keyword Reference

See Also: MessageBox, GetEasyChoice, EasyTextBox

Name/Symbol: GetEasyInput()

Description: Hides what is on the screen and presents a textbox with specified message, and a second textbox to enter input. Continues when 'enter' is hit at the end of text entry.

Usage:

GetEasyInput(<message>,<window>)

Example:

```r
gWin <- MakeWindow()
inp <- GetEasyInput("Enter Participant ID Code",gWin)
```

See Also: MessageBox(), GetEasyChoice(), EasyTextBox()

Name/Symbol: GetInput()

Description: Allows user to type input into a textbox.

Usage: GetInput(<textbox>,<escape-key>)

Example:

See Also: SetEditable(), GetCursorPosition(), MakeTextBox(), SetText()

Name/Symbol: GetJoystickAxisState
Chapter 8. Detailed Function and Keyword Reference

Description: This gets the state of a particular joystick axis. You need to specify a joystick object, which is created with OpenJoystick(). You also need to specify the axis. You can determine how many axes a joystick has with the GetNumJoystickAxes() function. The function returns a value between 1 and 32768.

Usage: \texttt{GetJoystickAxisState(js,1)}

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats() GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState() GetJoystickBallState)

Name/Symbol: GetJoystickButtonState

Description: This gets the state of a particular joystick button. You need to specify a joystick object, which is created with OpenJoystick(). You also need to specify the button. You can determine how many buttons a joystick has with the GetNumJoystickButtons() function. The function returns either 0 (for unpressed) or 1 (for pressed).

Usage: \texttt{GetJoystickButtonState(js,1)}

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats() GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState() GetJoystickBallState)

Name/Symbol: GetJoystickBallState

Description: Not implemented.

Usage: \texttt{GetJoystickBallState(js,1)}

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats() GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState() GetJoystickBallState)
Name/Symbol: GetJoystickHatState

Description: GetJoystickHatState(js,1) This gets the state of a particular joystick hat. You need to specify a joystick object, which is created with OpenJoystick(). You also need to specify the hat id. You can determine how many hats a joystick has with the GetNumJoystickHats() function. The function returns a value between 0 and 15, which is the sum of values specifying whether each primary NSEW direction is pressed. The coding is: 0=no buttons; 1=N, 2=E, 4=S, 8=W. Thus, if 1 is returned, the north hat button is pressed. If 3 is returned, NorthEast. If 12 is returned, SW, and so on.

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(), GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats(), GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState()
Example:  

```plaintext
define Start(p) {

    win <- MakeWindow()
    i <- 1
    while(i < 100) {
        Draw()
        Print(GetMouseState())
        Wait(100)
        i <- i + 1
    }
    ##Returns look like:
    [417, 276, 0, 0, 0]
    [495, 286, 0, 0, 0]
    [460, 299, 0, 0, 0]
    [428, 217, 0, 0, 0]
    [446, 202, 0, 0, 4]
    [446, 202, 1, 0, 0]
    [446, 202, 1, 0, 0]
    [446, 202, 0, 2, 0]
}

See Also:  ShowCursor WaitForMouseButton, SetMouseCursorPosition, GetMouseCursorPosition
```

Name/Symbol: GetNewDataFile()  
Description:  Creates a data file for output, asking for either append or renumbering the subject code if the specified file is already in use.  

Usage:  

```plaintext
GetNewDataFile(subnum,win,basename,extension,header)
```

Here, subnum should be a subject code you want to use. win should refer to the window a prompt will be displayed on if the subject code is already in use. basename should be the base filename of the file, and extension should be the .extension (without the dot) at the end of the file. Finally, header is what will be printed on the first row of a file.

When this file is used, a 'data' subdirectory will first be created in the current directory (i.e., the directory where the experiment is). Then, a subdirectory will be created inside data based on the subnum. Spaces and some other characters will be removed.
to ensure easy and uniform access to this directory. Then, a filename will be created composed of:

\texttt{data\subnum\basename-subnum.extension}

If this file does not exist, one will be created and the header will be printed to the first line. If it does exist, you will be prompted that the file exists, and at that point you can choose to either append to the existing file (in which case no header will be added), or choose a new subject code (in which case, a new directory will be made). The process can repeat until you either append or choose an unused file.

Multiple files can be made, and they will all appear in the subnum directory. If you get a filename collision, your decision on the first file will carry forward on future files, controlled by a special global variable called \texttt{gResetNumber}.

Note that nearly all of the test battery tests use this function. This can make pooling subject data more difficult, but use the combine data dialog in the launcher to easily combine data from multiple files in multiple subdirectories.

In all situations, the global variable \texttt{gSubNum} is set to the resulting subject code (whether or not \texttt{gSubNum} is passed to this function). This should only be called at the beginning of an experiment, when the experimenter still has control of the computer, in case a subject code is reused and a decision needs to be made. Finally, the extension chosen has essentially nothing to do with how the internals are formatted; it is up to you.

Example:

\begin{verbatim}
file1 <- GetNewDataFile("1",gWin,"memorytest","csv",
                      "sub,trial,word,answer,rt,corr")
## above creates a file data\1\memorytest-1.csv

file2 <- GetNewDataFile("1",gWin,"memorytest","csv",
                      "sub,trial,word,answer,rt,corr")
# above will prompt you for new subject code

file3 <- GetNewDataFile("1",gWin,"memorytest-report","txt",
                        "")
## No header is needed on a text-based report file.
\end{verbatim}

See Also: \texttt{FileOpenWrite}, \texttt{FileOpenAppend}, \texttt{FileOpenOverwrite}

Name/Symbol: \texttt{GetNIMHDemographics}()
Chapter 8. Detailed Function and Keyword Reference

Description: Gets demographic information that are normally required for NIMH-related research. Currently are gender (M/F/prefer not to say), ethnicity (Hispanic or not), and race (A.I./Alaskan, Asian/A.A., Hawaiian, black/A.A., white/Caucasian, other). It then prints their responses in a single line in the demographics file, along with any special code you supply and a time/date stamp. This code might include a subject number, experiment number, or something else, but many informed consent forms assure the subject that this information cannot be tied back to them or their data, so be careful about what you record. The file output will look something like:

```
----
31, Thu May 12 17:00:35 2011, F, hisp, asian, 3331
32, Thu May 12 22:49:10 2011, M, nothisp, amind, 3332
----
```

The first column is the user-specified code (in this case, indicating the experiment number). The middle columns indicate date/time, and the last three columns indicate gender (M, F, other), Hispanic (Y/N), and race.

Usage: 

```
GetNIMHDemographics(<code-to-print-out>,
                    <window>, <filename>)
```

Example: 

```
GetNIMHDemographics("x0413", gwindow,
                     "x0413-demographics.txt")
```

See Also:

Name/Symbol: GetNumJoystickAxes

Description: This gets the number of axes on a joystick. You need to specify a joystick object, which is created with OpenJoystick().

Usage: 

```
GetNumJoystickAxes(js, 1)
```

Example: 

```
See joysticktest.pbl in the demo directory
```

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(), GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats(), GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState()
Chapter 8. Detailed Function and Keyword Reference

Description: This gets the number of joystick balls available on a particular joystick. You need to specify a joystick object, which is created with OpenJoystick().

Usage: GetNumJoystickBalls(js)

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(), GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats() GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState()

Name/Symbol: GetNumJoystickButtons

Description: This gets the number of joystick buttons available on a particular joystick. You need to specify a joystick object, which is created with OpenJoystick().

Usage: GetNumJoystickButtons(js,1)

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(), GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats() GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState()

Name/Symbol: GetNumJoystickHats

Description: This gets the number of hats available on a particular joystick. You need to specify a joystick object, which is created with OpenJoystick().

Usage: GetNumJoystickHats(js,1)

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(), GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats() GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState()

Name/Symbol: GetNumJoysticks

137
Chapter 8. Detailed Function and Keyword Reference

Description: This gets the number of joysticks available on a system. It returns an integer, which if greater than you can open a joystick using the OpenJoystick() function.

Usage: GetNumJoysticks()

Example: See joysticktest.pbl in the demo directory

See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes(), GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats(), GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonState()

Name/Symbol: GetPEBLVersion()

Description: Returns a string describing which version of PEBL you are running.

Usage: GetPEBLVersion()

Example: Print(GetPEBLVersion())

See Also: TimeStamp()

Name/Symbol: GetPixelColor()

Description: Gets a color object specifying the color of a particular pixel on a widget.

Usage: color <- GetPixelColor(widget,x,y)

Example: 
##Judge brightness of a pixel
img <- MakeImage("test.png")
col <- GetPixelColor(img,20,20)
hsv <- RGBtoHSV(col)
Print(Third(hsv))

See Also: SetPixel()

Name/Symbol: GetPPortState

Description: Gets the parallel port state, as a list of 8 'bits' (1s or 0s).

Usage: out <- SetPPortState(pport)
Chapter 8. Detailed Function and Keyword Reference

Example:

See Also: COMPortGetByte, COMPortSendByte, OpenPPort OpenCOMPort, SetPPortMode, GetPPortState

Name/Symbol: GetProperty

Description: Gets a particular named property of an object. This works for custom or built-in objects. If the property does not exist, a fatal error will be signaled, and so you should check using PropertyExists() if there is any chance the property does not exist.

Usage:

```
out <- GetProperty(obj, property)
```

Example:

```
obj <- MakeCustomObject("myobject")
obj.taste <- "buttery"
obj.texture <- "creamy"
setProperty(obj,"flavor","tasty")

list <- GetPropertyList(obj)
loop(i,list)
{
  if(PropertyExists(obj,i))
  {
    Print(i + ": " + GetProperty(obj,i))
  }
}
```

See Also: GetPropertyList, PropertyExists, SetProperty

Name/Symbol: GetPropertyList

Description: Gets a list of all of the properties an object has. This works for custom or built-in objects.

Usage:

```
out <- GetPropertyList(obj)
```

Example:

```
obj <- MakeCustomObject("myobject")
obj.taste <- "buttery"
obj.texture <- "creamy"
setProperty(obj,"flavor","tasty")
```
list <- GetPropertyList(obj)
loop(i,list)
{
    if(PropertyExists(obj,i))
    {
        Print(i + ": " + GetProperty(obj,i))
    }
}

See Also: GetProperty, PropertyExists, SetProperty
MakeCustomObject, PrintProperties

Name/Symbol:GetSize()
Description: Returns a list of [height, width], specifying the size of the widget. The .width and .height properties can also be used instead of this function
Usage: GetSize(<widget>)
Example: image <- MakeImage("stim1.bmp")
xy <- GetSize(image)
x <- Nth(xy, 1)
y <- Nth(xy, 2)

See Also:

Name/Symbol:GetSubNum()
Description: Creates dialog to ask user to input a subject code
Usage: GetSubNum(<win>)
Example: ## Put this at the beginning of an experiment,
## after a window gWin has been defined.
##
## if(gSubNum == 0)
## {
##    gSubNum <- GetSubNum(gWin)
## }  
Note: gSubNum can also be set from the command line.
See Also:
Name/Symbol: GetSystemType()
Description: Returns a string identify what type of computer system you are using. It will return either: OSX, LINUX, or WINDOWS.
Usage: GetSystemType()
Example: ## Put this at the beginning of an experiment, ## after a window gWin has been defined.  
        if(GetSystemType() == "WINDOWS")  
        {  
            SignalFatalError("Experiment untested on windows")  
        }
See Also: SystemCall()

Name/Symbol: GetText()
Description: Returns the text stored in a text object (either a textbox or a label). The .text properties can also be used instead of this function.
Usage: GetText(<widget>)
Example: See Also: SetCursorPosition(), GetCursorPosition(), SetEditable(), MakeTextBox()

Name/Symbol: GetTextBoxCursorFromClick()
Description: Returns the position (in characters) corresponding to a x,y click on a text box. The X,Y position must be relative to the x,y position of the box, not absolute. Once obtained, the cursor position can be set with SetCursorPosition().
Usage: GetTextBoxCursorFromClick(<widget>,<x>,<y>)
Example: win <- MakeWindow()  
tb <- EasyTextBox("Click here to set cursor position"  
,100,100,win,200,200)  
Draw()  
WaitForClickOnTarget([tb],[[1]])  
# get the x and y cursor positions  
relx <- First(gClick) - (tb.x)  
rely <- Second(gClick) - (tb.y)
Chapter 8. Detailed Function and Keyword Reference

```r
tb.cursorpos <- GetTextBoxCursorFromClick(tb, relx, rely)
Draw()
WaitForAnyKeyPress()
```

See Also: `SetCursorPosition()`, `GetCursorPosition()`, `SetEditable()`, `MakeTextBox()`

---

Name/Symbol: `GetTime()`

Description: Gets time, in milliseconds, from when PEBL was initialized. Do not use as a seed for the RNG, because it will tend to be about the same on each run. Instead, use `RandomizeTimer()`.

Usage: `GetTime()`

Example:
```
a <- GetTime()
WaitForKeyDown("A")
b <- GetTime()
Print("Response time is: " + (b - a))
```

See Also: `TimeStamp()`

---

Name/Symbol: `GetVideoModes()`

Description: Gets a list of usable video modes (in width/height pixel pairs), as supplied by the video driver.

Usage: `modes <- GetVideoModes()`

Example:
```
Print(GetVideoModes)
## Might return:
[[1440, 900],
 [1360, 768],
 [1152, 864],
 [1024, 768],
 [960, 600],
 [960, 540],
 [840, 525],
 [832, 624],
 [800, 600],
 [800, 512],
 [720, 450],
 [720, 400],
 [700, 525]]
```
Name/Symbol: GetVocalResponseTime

Description: This is a simple audio amplitude voice key controlled by two parameters ONLY AVAILABLE ON WINDOWS AND LINUX.

Usage: GetVocalResponseTime(buffer, timethreshold, energythreshold)

This is a simple function that fairly reliably gets an audio response time. It works by recording audio to a buffer, and computing energy for 1-ms bins. When enough bins (whose number/duration is set by timethreshold) in a row surpass an energy threshold (scaled from 0 to 1, set by energythreshold), recording will stop, and the voice key will return. Reasonable values depend on the amount of noise in your microphone, and the types of vocal responses being made. The return time will lag the detection time a bit, and so using the time it takes for the function to return is an unreliable measure of vocal response time.

It returns a list of three elements:

- Response time (in ms),
- End time (using ms counter),
- Responded flag: either 0 or 1, depending on whether the key was tripped.

If the responded flag is 0, the other two numbers will be as well.

See number-stroop.pbl in the stroop directory of the test battery and testaudioin.pbl in demo/ for examples.

Example:

```r
buffer <- MakeAudioInputBuffer(5000)
resp0 <- GetVocalResponseTime(buffer,.35, 200)
SaveAudioToWaveFile("output.wav",buffer)
```

See Also: MakeAudioInputBuffer(), SaveAudioToWaveFile(),
8.9 H

Name/Symbol: Hide()

Description: Makes an object invisible, so it will not be drawn.

Usage: Hide(<object>)

Example:
window <- MakeWindow()
image1 <- MakeImage("pebl.bmp")
image2 <- MakeImage("pebl.bmp")
AddObject(image1, window)
AddObject(image2, window)
Hide(image1)
Hide(image2)
Draw() # empty screen will be drawn.
Wait(3000)
Show(image2)
Draw() # image2 will appear.

Hide(image2)
Draw() # image2 will disappear.

Wait(1000)
Show(image1)
Draw() # image1 will appear.

See Also: Show()
Chapter 8. Detailed Function and Keyword Reference

8.10 I

Name/Symbol: if

Description: Simple conditional test.

Usage:
```
if(test)
{
    statements
to
ever be
executed
}
```

Example:
```
See Also:
```

Name/Symbol: if...elseif...else

Description: Complex conditional test. Be careful of spacing the else—if you
put carriage returns on either side of it, you will get a syntax
error. The elseif is optional, but multiple elseif statements
can be strung together. The else is also optional, although
only one can appear.

Usage:
```
if(test)
{
    statements if true
} elseif (newtest) {
    statements if newtest true; test false
} else {
    other statements
}
```

Example:
```
if(3 == 1) {
    Print("ONE")
} elseif(3==4){
    Print("TWO")
} elseif(4==4){
    Print("THREE")
} elseif(4==4){
    Print("FOUR")
} else{Print("FIVE")}
```

See Also: if

145
Name/Symbol: Insert()
Description: Inserts an element into a list at a specified position, returning the new list. The original list in unchanged.
Usage: Insert(<[list]>,<item>,<position>)
Example:
\[
\begin{align*}
x & \leftarrow [1,2,3,5] \\
y & \leftarrow \text{Insert}(x,1,4) \\
\text{##y} & \leftarrow [1,2,3,1,5]
\end{align*}
\]
See Also: List(), Merge, Append

Name/Symbol: Inside()
Description: Determines whether an \([x,y]\) point is inside another object. Will operate correctly for rectangles, squares, circles, textboxes, images, and labels. For custom objects having a function name bound to their .inside property, it will use that function to test for insideness. \([xylist]\) can be a list containing \([x,y]\), and if it is longer the other points will be ignored (such as the list returned by WaitForMouseButton()). Returns 1 if inside, 0 if not inside.
Usage: Inside(<[xylist]>,<object>)
Example:
\[
\begin{align*}
\text{button} & \leftarrow \text{EasyLabel}("\text{Click me to continue}", 100,100,g\text{Win},12) \\
\text{continue} & \leftarrow 1 \\
\text{while(continue)} \\
\{ \\
\text{xy} & \leftarrow \text{WaitForMouseButton()} \\
\text{continue} & \leftarrow \text{Inside(xy,button)} \\
\}
\end{align*}
\]
See Also: WaitForMouseButton(), GetMouseCursorPosition, InsideTB

Name/Symbol: InsideTB()
Description: Determines whether an \([x,y]\) point is inside an object having .x, .y, .width, and .height properties, with .x and .y representing the upper left corner of the object. This is bound to the .inside property of many custom ui objects. The Inside function will use the function bound to the .inside property for any custom object having that property, and so this function’s use is mainly hidden from users.
Chapter 8. Detailed Function and Keyword Reference

Usage: \[\text{InsideTB}([x,y], \text{<obj>})\]

Example:
\[
\text{pulldown} \leftarrow \text{MakePulldown}(\text{"one","two","three","four"}, 400-75, 300, \text{gWin}, 12, 150, 1)
\]
\[
\text{if(InsideTB}([300,300], \text{pulldown})\) \\
\{ \\
\text{\hspace{1em}Print("INSIDE")}
\}
\]

See Also: Inside(), MoveObject, ClickOn, DrawObject

Name/Symbol: \text{IsAnyKeyDown()} \\
Description: \text{\hspace{1em}Tests whether <variant> is a key down.} \\
Usage: \text{IsAnyKeyDown()} \\
Example: 

See Also: IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox(), IsWidget()

Name/Symbol: \text{IsAudioOut()} \\
Description: \text{\hspace{1em}Tests whether <variant> is an AudioOut stream.} \\
Usage: \text{IsAudioOut(<variant>)} \\
Example: 
\[
\text{if(IsAudioOut(x))} \\
\{ \\
\text{\hspace{1em}Play(x)} \\
\}
\]

See Also: IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox(), IsWidget()

Name/Symbol: \text{IsCanvas()} \\
Description: \text{\hspace{1em}Tests whether <variant> is a Canvas widget.} \\
Usage: \text{IsCanvas(<variant>)} \\
Example: 
\[
\text{if(IsCanvas(x)} \\
\{ \\
\text{\hspace{1em}SetPixel(x,10,10,MakeColor("red"))} \\
\}
\]

147
See Also: IsAudioOut(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumberO(), IsString(), IsTextBox(), IsText() IsWidget(), IsWindow()

Name/ Symbol: IsColor()

Description: Tests whether <variant> is a Color.
Usage: IsColor(<variant>)
Example: if(IsColor(x)
   {gWin <- MakeWindow(x)
    }

See Also: IsAudioOut(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumberO(), IsString(), IsTextBox(), IsText() IsWidget(), IsWindow()

Name/ Symbol: IsCustomObject()

Description: Tests whether <variant> is a Custom object (created with MakeCustomObject.) Return 1 if so, 0 if not.
Usage: IsCustomObject(<obj>)
Example: if(IsCustomObject(obj)
   {MoveObject(obj,x,y)
    } else {
        Move(obj,x,y)
    }

See Also: IsAudioOut(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumberO(), IsString(), IsTextBox(), IsText() IsWidget(), IsWindow()

Name/ Symbol: IsDirectory()

Description: Determines whether a named path is a directory. Returns 1 if it exists and is a directory, and 0 otherwise.
Usage: IsDirectory(<path>)
Example:

```r
filename <- "data-"+gSubNum+".csv"
exists <- FileExists(filename)
if(exists)
{
  out <- IsDirectory(filename)
  Print(out)
}
```

See Also: `GetDirectoryListing()`, `FileExists()`, `IsDirectory()`, `MakeDirectory()`

Name/Symbol: `IsImage()`

Description: Tests whether `<variant>` is an Image.

Usage: `IsImage(<variant>)`

Example:

```r
if(IsImage(x))
{
  AddObject(gWin, x)
}
```


Name/Symbol: `IsInteger()`

Description: Tests whether `<variant>` is an integer type. Note: a number represented internally as a floating-point type whose is an integer will return false. Floating-point numbers can be converted to internally-represented integers with the `ToInteger()` or `Round()` commands.

Usage: `IsInteger(<variant>)`

Example:

```r
x <- 44
y <- 23.5
z <- 6.5
test <- x + y + z
IsInteger(x) # true
IsInteger(y) # false
IsInteger(z) # false
IsInteger(test) # false
```
Chapter 8. Detailed Function and Keyword Reference

See Also: IsAudioOut(), IsColor(), IsImage(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox(), IsWidget()

Name/Symbol: IsFileStream()
Description: Tests whether <variant> is a FileStream object.
Usage: IsFileStream(<variant>)
Example: if(IsFileStream(x))
{ 
  Print(FileReadWord(x))
}
See Also: IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox(), IsWidget()

Name/Symbol: IsFloat()
Description: Tests whether <variant> is a floating-point value. Note that floating-point can represent integers with great precision, so that a number appearing as an integer can still be a float.
Usage: IsFloat(<variant>)
Example: x <- 44
       y <- 23.5
       z <- 6.5
       test <- x + y + z

       IsFloat(x) # false
       IsFloat(y) # true
       IsFloat(z) # true
       IsFloat(test) # true
See Also: IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox(), IsWidget()

Name/Symbol: IsFont()
Description: Tests whether <variant> is a Font object.
Chapter 8. Detailed Function and Keyword Reference

Usage: \texttt{IsFont(<variant>)}

Example: \begin{verbatim}
  if(IsFont(x))
  {
    y <- MakeLabel("stimulus", x)
  }
\end{verbatim}

See Also: \texttt{IsAudioOut()}, \texttt{IsColor()}, \texttt{IsImage()}, \texttt{IsInteger()}, \texttt{IsFileStream()}, \texttt{IsFloat()}, \texttt{IsLabel()}, \texttt{IsList()}, \texttt{IsNumber()}, \texttt{IsString()}, \texttt{IsTextBox()}, \texttt{IsWidget()}

Name/Symbol: \texttt{IsKeyDown()}

Description:

Usage:

Example:

See Also: \texttt{IsKeyUp()}

Name/Symbol: \texttt{IsKeyUp()}

Description:

Usage:

Example:

See Also: \texttt{IsKeyDown()}

Name/Symbol: \texttt{IsLabel()}

Description: Tests whether \texttt{<variant>} is a text Label object.

Usage: \texttt{IsLabel(<variant>)}

Example: \begin{verbatim}
  if(IsLabel(x))
  {
    text <- GetText(x)
  }
\end{verbatim}

See Also: \texttt{IsAudioOut()}, \texttt{IsColor()}, \texttt{IsImage()}, \texttt{IsInteger()}, \texttt{IsFileStream()}, \texttt{IsFloat()}, \texttt{IsFont()}, \texttt{IsList()}, \texttt{IsNumber()}, \texttt{IsString()}, \texttt{IsTextBox()}, \texttt{IsWidget()}

151
Name/Symbol: `IsList()`

Description: Tests whether `<variant>` is a PEBL list.

Usage: `IsList(<variant>)`

Example: `if(IsList(x))
{
    loop(item, x)
    {
        Print(item)
    }
}
`


---

Name/Symbol: `IsMember()`

Description: Returns true if `<element>` is a member of `<list>`.

Usage: `IsMember(<element>,<list>)`

Example: `IsMember(2,[1,4,6,7,7,7,7]) # false
IsMember(2,[1,4,6,7,2,7,7,7]) # true`

See Also:

---

Name/Symbol: `IsNumber()`

Description: Tests whether `<variant>` is a number, either a floating-point or an integer.

Usage: `IsNumber(<variant>)`

Example: `if(IsNumber(x))
{
    Print(Sequence(x, x+10, 1))
}
`

Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: IsShape

Description: Tests whether <variant> is a drawable shape, such as a circle, square rectangle, line, bezier curve, or polygon.

Usage: IsShape(<variant>)

Example:
if(IsShape(x))
{
   Move(x,300,300)
}

See Also: Square(), Circle(), Rectangle(), Line(), Bezier(), Polygon() IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox(), IsWindow()

Name/Symbol: IsString()

Description: Tests whether <variant> is a text string.

Usage: IsString(<variant>)

Example:
if(IsString(x))
{
   tb <- MakeTextBox(x, 100, 100)
}

See Also: IsText() IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox(), IsWidget()

Name/Symbol: IsText()

Description: Tests whether <variant> is a text string. Same as IsString().

Usage: IsString(<variant>)

Example:
if(IsText(x))
{
   tb <- MakeTextBox(x, 100, 100)
}
Chapter 8. Detailed Function and Keyword Reference

See Also: IsString(), IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsTextBox(), IsWidget()

Name/Symbol: IsTextBox()
Description: Tests whether <variant> is a TextBox Object
Usage: IsTextBox(<variant>)
Example:
```
if(IsTextBox(x))
{
    Print(GetText(x))
}
```
See Also: IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsWidget()

Name/Symbol: IsWidget
Description: Tests whether <variant> is any kind of a widget object (image, label, or textbox).
Usage: IsWidget(<variant>)
Example:
```
if(IsWidget(x))
{
    Move(x, 200, 300)
}
```
See Also: IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox()

Name/Symbol: IsWindow
Description: Tests whether <variant> is a window.
Usage: IsWindow(<variant>)
Example:
```
if(IsWindow(x))
{
    AddObject(y, x)
}
```
See Also: IsAudioOut(), IsColor(), IsImage(), IsInteger(), IsFileStream(), IsFloat(), IsFont(), IsLabel(), IsList(), IsNumber(), IsString(), IsTextBox()
Chapter 8. Detailed Function and Keyword Reference

8.11 K

Name/Symbol: KaniszaPolygon

Description: Creates generic polygon, defined only by with “pac-man” circles at specified vertices.

Usage: 

\[
\text{KaniszaPolygon}(\text{xypoints}, \text{vertices-to-show}, \text{circle-size}, \text{fgcol}, \text{bgcol}, \text{show-edge})
\]

Example: For detailed usage example, see:


Part of a script using KaniszaPolygon:

```r
#Specify the xy points
xys <- \[[10,10], [10,50], [130,60], [100,100], [150,100],
         [150,20], [80,-10], [45,10]\]

#Specify which vertices to show (do all)
show <- \[1,1,1,1,1,1,1,1\]

#Make one, showing the line
x <- KaniszaPolygon(xys, show, 10, fg, bg, 1)
AddObject(x,gWin); Move(x,200,200)

#Make a second, not showing the line
x2 <- KaniszaPolygon(xys, show, 10, fg, bg, 0)
AddObject(x2,gWin); Move(x2,400,200)

#Make a third, only showing some vertices:
x3 <- KaniszaPolygon(xys, [1,1,1,1,0,0,1], 10, fg, bg, 0)
AddObject(x3,gWin); Move(x3,600,200)
```

See Also: Polygon(), KaniszaSquare()
Name/Symbol: KaniszaSquare

Description: Creates generic Kanisza Square, one defined only by with “pacman” circles at its vertices:

Usage: KaniszaSquare(<size>, <circ-rad>, <fgcol>, <bgcol>)

KaniszaSquare creates a graphical object that can be added to a window, moved to the proper location, etc. Parameters specify the size of the square, the size of the vertex circles, and the foreground and background colors.

Example: For detailed usage example, see http://peblblog.blogspot.com/2010/11/kanizsa-shapes.html

```r
  gWin <- MakeWindow()
  square <- KaniszaSquare(150,20,MakeColor("red"), MakeColor("green"))
  AddObject(square,gWin)
  Move(square,200,200)
  Draw()
  WaitForAnyKeyPress()
```

See Also: Polygon(), KaniszaPolygon()
8.12 L

Name/Symbol: Last()
Description: Returns the last item in a list. Provides faster access to the last item of a list than does Nth().
Usage: Last(<list>)
Example: Last([1,2,3,444]) # == 444
See Also: Nth(), First()

Name/Symbol: LatinSquare()
Description: Quick and dirty latin square, taking on just one list argument.
Usage: LatinSquare(<list>)
Example: Print(LatinSquare([11,12,13,14,15,16]))
  # Output:
  # [[11, 12, 13, 14, 15, 16]
  #, [12, 13, 14, 15, 16, 11]
  #, [13, 14, 15, 16, 11, 12]
  #, [14, 15, 16, 11, 12, 13]
  #, [15, 16, 11, 12, 13, 14]
  #, [16, 11, 12, 13, 14, 15]
  #]
See Also: DesignFullCounterBalance(), DesignBalancedSampling(), DesignGrecoLatinSquare(), DesignLatinSquare(), Repeat(), RepeatList(), Shuffle()

Name/Symbol: LaunchFile()
Description: Launch a specified file or URI with a platform-specific handler.
Usage: LaunchFile("filename")
Example: Example uses:


Chapter 8. Detailed Function and Keyword Reference

#open google:
LaunchFile("http://google.com")

#Open a .pbl file with text editor:
LaunchFile("test.pbl")

#Open a data directory in file manager:
LaunchFile("data")

See Also: SystemCall()

Name/Symbol: LayoutGrid

Description: Creates a grid of x,y points in a range, that are spaced in a specified number of rows and columns. Furthermore, you can specify whether they are vertical or horizontally laid out.

Usage: LayoutGrid(<xmin>,<xmax>,<ymin>,<ymax>,<columns>,<rows>,<vertical>)

Example: Example PEBL Program using NonoverlapLayout:

```pbl
define Start(p)
{
    gWin <- MakeWindow()
gVideoWidth <- 800
gVideoHeight <- 300

    lab1 <- EasyLabel("LayoutGrid, horizontal",
                      200,25,gWin,24)
    lab2 <- EasyLabel("LayoutGrid, vertical",
                      600,25,gWin,24)

    nums <- Sequence(1,20,1)
stim1 <- []
stim2 <- []

    font <- MakeFont(gPeblBaseFont,0,25,
                      MakeColor("black"),MakeColor("white"),0)
    loop(i,nums)
    {
        stim1 <- Append(stim1,MakeLabel(i"",font))
        stim2 <- Append(stim2,MakeLabel(i"",font))
    }

    layout1 <- LayoutGrid(50,gVideoWidth/2-50,
                           50,gVideoHeight-50,5,4,0)
    layout2 <- LayoutGrid(gVideoWidth/2+50,gVideoWidth-50,
                           gVideoHeight-50,5,4,0)
}
```
###Now, layout the stuff.

```r
loop(i, Transpose([stim1, layout1]))
{
  obj <- First(i)
  xy <- Second(i)
  AddObject(obj, gWin)
  Move(obj, First(xy), Second(xy))
}
```

```r
loop(i, Transpose([stim2, layout2]))
{
  obj <- First(i)
  xy <- Second(i)
  AddObject(obj, gWin)
  Move(obj, First(xy), Second(xy))
}
```

Draw()
WaitForAnyKeyPress()
}
```

The output of the above program is shown below. Even for the left configuration, which is too compact (and which takes a couple seconds to run), the targets are fairly well distributed.
Chapter 8. Detailed Function and Keyword Reference

See Also: NonOverlapLayout()

Name/Symbol: Line()
Description: Creates a line for graphing at x, y ending at x+dx, y+dy. dx and dy describe the size of the line. Lines must be added to a parent widget before it can be drawn; it may be added to widgets other than a base window. Properties of lines may be accessed and set later.
Usage: Line(<x>, <y>, <dx>, <dy>, <color>)
Example:
```lisp
l <- Line(30, 30, 20, 20, MakeColor("green"))
AddObject(l, win)
Draw()
```
See Also: Square(), Ellipse(), Rectangle(), Circle()

Name/Symbol: List()
Description: Creates a list of items. Functional version of [].
Usage: List(<item1>, <item2>, ....)
Example:
```lisp
List(1,2,3,444) # == [1,2,3,444]
```
See Also: [], Merge(), Append()

Name/Symbol: ListBy()
Description: organizes a list into sublists, based on the elements of a second list. It returns a list of two entities: (1) a condition list, describing what values were aggregated across; (2) the nested list elements. The length of each element should be the same. Together with Match and Filter, ListBy is useful for aggregating data across blocks and conditions for immediate feedback.
Usage: ListBy(<list>, <conds>)
Example:
```lisp
a <- Sequence(1,10,1)
b <- RepeatList([1,2],5)
x <- ListBy(a,b)
Print(x)
```
Chapter 8. Detailed Function and Keyword Reference

```
# [[1, 2],
#  [1, 3, 5, 7, 9],
#  [2, 4, 6, 8, 10]]
#
Print(ListBy(b,a))
# [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
#  [[1], [2], [1], [2], [1], [2], [1], [2], [1], [2]]]
```

See Also: List(), [ ], Merge(), Append()

Name/Symbol: ListToString()
Description: Converts a list of things to a single string
Usage: ListToString(<list>)
Example: ListToString([1,2,3,444]) # == "123444"
        ListToString(\["a","b","c","d","e"\]) # == "abcde"
See Also: SubString, StringLength

Name/Symbol: Length()
Description: Returns the number of items in a list.
Usage: Length(<list>)
Example: Length([1,3,55,1515]) # == 4
See Also: StringLength()

Name/Symbol: Levels()
Description: Returns sorted list of unique elements of a list.
Usage: Levels(<list>)
Example: Levels([1,3,55,1,5,1,5]) # == [1,3,5,55]
See Also: Match(), Filter(), Sort()
Name/Symbol: LoadAudioFile()

Description: Loads an audio file supported by the ffmpeg library. It is nearly identical to LoadMovie(), but only works for audio files (.ogg, .mp3, .wav, .aiff, .wma, et.). It creates a movie object, which can then be played using PlayMovie() or StartPlayback() functions. Currently, only supported on Windows and Linux.

The ffmpeg (http://ffmpeg.org) library supports a wide range of audio formats, including most .wav, .mp3, .ogg, .flac, .aiff, .wma, and others. Currently, there appears to sometimes be playback problems if the audio stream is not stereo, so be sure to convert your audio to stereo. Also, there appears to be some problems with .flac data formats.

If you have problems with playback, you should verify that your media file loads with another ffmpeg media player.

Usage: LoadAudioFile(audiofile)

Example:

```r
movie <- LoadAudioFile("instructions.mp3")
PrintProperties(inst)
PlayMovie(inst)
PausePlayback(inst)
```

See Also: LoadMovie(), PlayMovie(), StartPlayback(), PausePlayback()

Name/Symbol: LoadMovie()

Description: Loads a movie file using the ffmpeg library. It creates a movie object, which can then be played using PlayMovie() or StartPlayback() functions. Currently, only supported on Windows and Linux.

The ffmpeg (http://ffmpeg.org) library supports a wide range of video and audio formats, including most .mpg, .avi, .ogg and .mp3 type formats. Audio-only formats should load and play with LoadMovie, but another function, LoadAudioFile(), has been created for these, as they do not need to be added to a window to work.

If you have problems with playback, you should verify that your media file loads with another ffmpeg media player.

For technical reasons, a movie MUST be loaded directly onto a window, and not another widget.

Usage: LoadMovie(movie, window, width, height)
Example:

```r
movie <- LoadMovie("movie.avi", gWin, 640, 480)
PrintProperties(movie)
Move(movie, 20, 20)
Draw()
StartPlayback(movie)
Wait(500) # Play 500 ms of the movie.
PausePlayback(movie)
```

See Also: `LoadAudioFile()`, `LoadMovie()`, `PlayMovie()`, `StartPlayback()`, `PausePlayback()`

---

**Name/Symbol:** `LoadSound()`

**Description:** Loads a soundfile from `<filename>`, returning a variable that can be played using the `PlayForeground` or `PlayBackground` functions. `LoadSound` only loads uncompressed .wav files, but uses a background mixer to play them with fairly low latency. In contrast, `LoadAudioFile` can load many different multimedia files other than .wav, and uses a different audio playback mechanism. `LoadSound` is appropriate for playing stimulus sounds and feedback, whereas `LoadAudioFile` may be more appropriate for instructions and longer feedback that should be encoded efficiently.

When the file gets loaded, it gets automatically transcoded into a stereo 44100-sampling rate audio stream, regardless of its original playback rate. We have reports that in some cases, this can cause some problems, especially if a mono file gets loaded multiple times in an experiment. If you experience playback problems, try converting your audio to stereo 44100 hz and see if it helps.

**Usage:**

```r
LoadSound(<filename>)
```

**Example:**

```r
woof <- LoadSound("dog.wav")
PlayBackground(woof)
Wait(200)
Stop(woof)
PlayForeground(woof)
```

See Also: `PlayForeground`, `PlayBackground`, `LoadAudioFile`, `LoadMovie`

---

**Name/Symbol:** `Log10()`

**Description:** Log base 10 of `<num>`.

164
Usage: \( \text{Log10}(<\text{num}>) \)

Example:

See Also: \( \text{Log2()}, \text{LogN()}, \text{Ln()}, \text{Exp()} \)

---

**Name/Symbol:** \( \text{Log2()} \)

**Description:** Log base 2 of \(<\text{num}>\).

**Usage:** \( \text{Log2}(<\text{num}>) \)

**Example:**

See Also: \( \text{Log()}, \text{Log2()}, \text{Ln()}, \text{Exp()} \)

---

**Name/Symbol:** \( \text{LogN()} \)

**Description:** Log base \(<\text{base}>\) of \(<\text{num}>\).

**Usage:** \( \text{LogN}(<\text{num}>, <\text{base}>) \)

**Example:**

See Also: \( \text{Log()}, \text{Log2()}, \text{Ln()}, \text{Exp()} \)

---

**Name/Symbol:** \texttt{Lowercase()}

**Description:** Changes a string to lowercase. Useful for testing user input against a stored value, to ensure case differences are not detected.

**Usage:** \texttt{Lowercase(<string>)}

**Example:** \texttt{Lowercase("POtaTo") \# == "potato"}

See Also: \texttt{Uppercase()}

---

**Name/Symbol:** \texttt{Ln()}

**Description:** Natural log of \(<\text{num}>\).

**Usage:** \texttt{Ln(<num>)}

---
**Chapter 8. Detailed Function and Keyword Reference**

Example:

See Also: Log(), Log2(), LogN(), Exp()

**Name/Symbol:** Lookup()

**Description:** Returns element in `<database>` corresponding to element of `<keylist>` that matches `<key>.

If no match exists, Match returns an empty list.

**Usage:**

```
Lookup(<key>,<keylist>,<database>)
```

**Example:**

```
keys <- [1,2,3,4,5]
database <- ["market","home","roast beef", "none","wee wee wee"]
Print(Lookup(3,keys,database))
```

## Or, do something like this:

```
data <- ["punky","brewster"],
["arnold","jackson"],
["richie","cunningham"],
["alex","keaton"]

d2 <- Transpose(data)
key <- First(data)

Print(Lookup("alex", key, data))
##Returns ["alex","keaton"]
```

See Also: Match

**Name/Symbol:** loop()

**Description:** Loops over elements in a list. During each iteration, `<counter>` is bound to each consecutive member of `<list>`.

**Usage:**

```
loop(<counter>, <list>)
{
  statements
to
be
executed
}
```
Chapter 8. Detailed Function and Keyword Reference

Example:

See Also: while(), { }
Name/Symbol: `MakeAttneave()`  
Description: Makes a random 'Attneave' figure\(^1\). An Attneave figure is a complex polygon that can be used as a stimulus in a number of situations. It returns a sequence of points for use in `Polygon()`.

MakeAttneave uses `ConvexHull`, `InsertAttneavePointRandom()` and `ValidateAttneaveShape()`, found in `Graphics.pbl`. Override these to change constraints such as minimum/maximum side lengths, angles, complexity, etc.

MakeAttneave uses a sampling-and-rejection scheme to create in-bounds shapes. Thus, if you specify impossible or nearly-impossible constraints, the time necessary to create shapes may be very long or infinite.

The arguments to MakeAttneave are:

- `size`: size, in pixels, of a circle from which points are sampled in a uniform distribution.
- `numpoints`: number of points in the polygon.
- `minangle`: smallest angle acceptable (in degrees).
- `maxangle`: largest angle acceptable (in degrees).

Usage: `MakeAttneave(size, numpoints, minangle, maxangle)`

**Chapter 8. Detailed Function and Keyword Reference**

Example:

```r
gWin <- MakeWindow()
shape <- MakeAttneave(100, 5+RandomDiscrete(5), 5, 170)
pts <- Transpose(shape)
poly <- Polygon(200, 200, First(pts), Second(pts),
                   MakeColor("blue"), 1)
AddObject(poly, gWin)
Draw()
WaitForAnyKeyPress()
```

See Also: `MakeImage()`, `Polygon()`, `Square()`

---

**Name/Symbol:** `MakeAudioInputBuffer(<time-in-ms>)`

**Description:** Creates a sound buffer to use for audio recording or voicekey sound input. It is currently very simple, allowing only to set the duration. By default, it record mono at 44100 hz.

**Usage:**

`MakeAudioInputBuffer(<time-in-ms>)`

See `number-stroop.pbl` in the stroop directory of the test battery for examples.

Note: Version 0.12 seems to have some trouble specifying buffers of different lengths. 5000 seems to work, but others (3500?) may not.

**Example:**

```r
buffer <- MakeAudioInputBuffer(5000)
resp0 <- GetVocalResponseTime(buffer, .35, 200)
SaveAudioToWaveFile("output.wav", buffer)
```

See Also: `GetVocalResponseTime()`, `SaveAudioToWaveFile()`

---

**Name/Symbol:** `MakeButton()`

**Description:** Creates a button on a window that can be clicked and launches actions. The button is always 20 pixels high (using images in media images), with a rounded grey background. The label text will be shrunk to fit the width, although this should be avoided as it can look strange. A button is a custom object made from images and text. It has a property 'clickon' that is bound to 'PushButton'.
A button will look like this:

![Test entry](Test entry) ![Test message box](Test message box)

Usage: \texttt{MakeButton(label, x, y, window, width)}

Example: The following creates a button, waits for you to click on it, and animates a button press

```
done <- MakeButton("QUIT", 400, 250, gWin, 150)
resp <- WaitForClickOnTarget([done], [1])
CallFunction(done.clickon, [done, gClick])
```

See Also: \texttt{PushButton()}, \texttt{MakeCheckBox()}

Name/Symbol: \texttt{MakeCanvas()}

Description: Makes a canvas object \(<x>\) pixels by \(<y>\) pixels, in color \(<\text{color}>\).

A canvas is an object that other objects can be attached to, and imprinted upon. When the canvas gets moved, the attached objects move as well. The background of a canvas can be made invisible by using a color with alpha channel \(== 0\). The SetPixel and SetPoint functions let you change individual pixels on a canvas, to enable adding noise, drawing functional images, etc. A canvas gets 'cleared' by calling \texttt{ResetCanvas(canvas)}\). Any object added to a canvas creates an 'imprint' on the canvas that remains if the object is moved. This allows you to use another image as a paintbrush on the canvas, and lets you to add noise to text. Because a text label gets re-rendered when its drawn, if you want to add pixel noise to a stimulus, you can create a label, add it to a canvas, then add pixel noise to the canvas.

Usage: \texttt{MakeCanvas(<x>, <y>, <\text{color}>)}

Example: \[
gWin <- MakeWindow()
clear <- MakeColor("white")
clear.alpha <- 0
#make a transparent canvas:
\]
Chapter 8. Detailed Function and Keyword Reference

```r
x <- MakeCanvas(300,300,clear)
AddObject(x,gWin)
Move(x,300,300)
img <- MakeImage("pebl.png")
AddObject(img,x)
Move(img,100,100)
Draw(x)  #imprint the image on the canvas
Move(img,100,200)
Draw(x)  #imprint the image on the canvas
Hide(img)

#draw a line on the canvas
i <- 10
red <- MakeColor("red")
while(i < 200)
{
    SetPixel(x,20,i,red)
    i <- i + 1
}
Draw()
WaitForAnyKeyPress()
```

See Also: MakeImage(), SetPixel(), MakeGabor(), ResetCanvas()

Name/Symbol: `MakeCheckbox()`

Description: Creates a checkbox on a window that can be clicked and keeps track of its status. The checkbox uses a MakeButton object as its base. The checkbox button is always 20 pixels high (using images in media images), with a rounded grey background. The label text will be shrunk to fit the width, although this should be avoided as it can look strange. It has a property 'clickon' that is bound to ClickCheckBox, which flips its state and updates the graphics. It has a property state which is either 0 or 1, depending on the state of the checkbox. Its initial state is 0. Its state can be set using the SetCheckBox() function.

A checkbox will look like this:

![Checkbox](image)

Usage: `MakeCheckBox(label,x,y,window,width)`
Example: The following creates a button, waits for you to click on it, and
animates a button press

    ok <- MakeCheckbox("OK?",400,250,gWin,150)
    resp <- WaitForClickOnTarget([ok],[1])
    CallFunction(done.clickon,[done,gClick])
    Draw()

Alternately:

    ok <- MakeCheckbox("OK?",400,250,gWin,150)
    resp <- WaitForClickOnTarget([ok],[1])
    ClickCheckBox(done,gClick)
    Draw()

Examples of its use can be found in demo
ui.pbl

See Also: ClickCheckBox(), SetCheckBox()

Name/Symbol: MakeColor()
Description: Makes a color from <colorname> such as “red”, “green”, and
nearly 800 others. Color names and corresponding RGB values
can be found in doc/colors.txt.
Usage: MakeColor(<colorname>)
Example: green <- MakeColor("green")
black <- MakeColor("black")
See Also: MakeColorRGB(), RGBtoHSV()

Name/Symbol: MakeColorRGB()
Description: Makes an RGB color by specifying <red>, <green>, and <blue>
values (between 0 and 255).
Usage: MakeColorRGB(<red>, <green>, <blue>)
Example: 
See Also: MakeColor(), RGBtoHSV()
Name/Symbol: MakeCustomObject

Description: Creates a 'custom' object that can encapsulate multiple properties. It takes a name as an argument, but this is currently not accessible.

Usage: 

```r
obj <- MakeCustomObject("mybutton")
```

Example: 

```r
obj <- MakeCustomObject("myobject")
obj.taste <- "buttery"
obj.texture <- "creamy"
SetProperty(obj, "flavor", "tasty")
```

```r
list <- GetPropertyList(obj)
loop(i, list)
{
  if(PropertyExists(obj, i))
  {
    Print(i + ": " + GetProperty(obj, i))
  }
}
```

See Also: GetPropertyList, PropertyExists, SetProperty, IsCustomObject, PrintProperties, GetProperty

---

Name/Symbol: MakeDirectory()

Description: Creates a directory with a particular name. It will have no effect of the directory already exists.

Usage: 

```r
FileExists(<path>)
```

Example: 

```r
# create data subdirectory + subject-specific directory
MakeDirectory("data")
MakeDirectory("data/" + gsubnum)
filename <- "data/" + gsubnum ++ "/output.csv"
```

See Also: GetDirectoryListing(), FileExists(), IsDirectory(), MakeDirectory()

---

Name/Symbol: MakeFont()
Chapter 8. Detailed Function and Keyword Reference

Description: Makes a font. The first argument must be a text name of a font. The font can reside anywhere in PEBL’s search path, which would primarily include the media/fonts directory, and the working directory (where the script is saved).

- style changes from normal to bold/underline, italic.
- fgcolor and bgcolor need to be colors, not just names of colors
- if show-backing is 0, the font gets rendered with an invisible background; otherwise with a bgcolor background. (Note: previous to PEBL 0.11, the final argument = 0 rendered the font with non anti-aliased background, which I can see almost no use for.)

Usage: MakeFont(<ttf_filename>, <style>, <size>, <fgcolor>, <bgcolor>, <show-backing>)

Example: font <- MakeFont("Vera.ttf", 0, 22, MakeColor("black"), MakeColor("white"), 1)

See Also:

Name/Symbol: MakeGabor()

Description: Creates a greyscale gabor patch, with seven variables:

- size (in pixels) of square the patch is drawn on
- freq: frequency of grating (number of wavelengths in size)
- sd: standard deviation, in pixels, of gaussian window
- angle: angle of rotation of grating, in radians
- phase: phase offset of grating (in radians)
- bglev: number between 0 and 255 indicating background color in greyscale.
Chapter 8. Detailed Function and Keyword Reference

Usage:  
\[ \text{MakeGabor}(\text{size}, \text{freq}, \text{sd}, \text{angle}, \text{phase}, \text{bglev}) \]

MakeGabor creates a canvas that can be used like any image. It must be added to the window, placed, and drawn to appear. Typically, it can take several seconds to create a patch of any large size, so it is usually best to create the gabor patches when the test is initialized, or save and load images using WritePNG().

Typically, a sd roughly 1/4 to 1/10 the size of size is necessary to avoid vignetting.

Example:
\begin{verbatim}
win <- MakeWindow()
patch <- MakeGabor(80, 0,10,0,0,100)
AddObject(patch,win)
Move(patch,200,200)
Draw()
\end{verbatim}

See Also: MakeAttneave(), SetPixel(), MakeCanvas()

Name/Symbol: MakeImage()

Description: Makes an image widget from an image file. .bmp formats should be supported; others may be as well.

Usage:  
\[ \text{MakeImage}(\text{filename}) \]

Example:

See Also:

Name/Symbol: MakeLabel()

Description: Makes a text label for display on-screen. Text will be on a single line, and the Move() command centers \texttt{text} on the specified point.

Usage:  
\[ \text{MakeLabel}(\text{text}, \text{font}) \]

Example:

See Also:

Name/Symbol: MakeMenu()
Chapter 8. Detailed Function and Keyword Reference

Description: Creates a menu containing multiple menu items, that automatically call functions specified by the command.

Usage: 

```
MakeMenu(label,x,y,window,fontsize, width, subitems,functions)
```

The subitems list should include the names of the menu options. The functions list should be the same length, and contain the function names called when one of those items is used. MakeMenu uses MakeMenuItem to create each one of those items. Menus can only be nested one-deep (no submenus allowed).

Example: This creates a menu and awaits clicking on. More complete examples are available in ui.pbl. It requires that MyMessage is created somewhere.

```
menu1 <- MakeMenu("File",0,0,gWin,14,10,
                   ["Open","Save","Save As","Quit"],
                   ["MYMESSAGE","MYMESSAGE","MYMESSAGE","MYMESSAGE"])

menu2<- MakeMenu("Edit",70,0,gWin,14,10,
                  ["Cut","Copy","Paste","Select"],
                  ["MYMESSAGE","MYMESSAGE","MYMESSAGE","MYMESSAGE"])

menu <- [menu1,menu2]
opt <- WaitForClickOntarget(menu,[1,2])
ClickOnMenu(Nth(menu,opt),gClick)
```

See Also: MakeMenuItem(), OpenSubMenus(), ClickOnMenu

Name/Symbol: MakeMenuItem()

Description: Creates a single menu containing a label, whose .click property is bound to some other function.
Chapter 8. Detailed Function and Keyword Reference

Usage: `MakeMenuItem(label,x,y,window,fontsize, width, function)`

This function is typically not used directly, but rather it is called via `MakeMenu`. However, it can be used as a quick-and-dirty button.

Example: This creates a menu and awaits clicking on. More complete examples are available in ui.pbl. It requires that MyMessage is created somewhere

```plaintext
menu1 <- MakeMenuItem("File",0,0,gWin,14,10,"MYMESSAGE")

menu2<- MakeMenu("Edit",70,0,gWin,14,10, "MYMESSAGE")

menus <- [menu1,menu2]
opt <- WaitForClickOntarget(menus,[1,2])
ClickOnMenu(Nth(menus,opt),gClick)
```

See Also: `MakeMenu()`, `OpenSubMenus()`, `ClickOnMenu`

Name/Symbol: `MakeNGonPoints()`

Description: Creates a set of points that form a regular n-gon. It can be transformed with functions like `RotatePoints`, or it can be used to create a graphical object with `Polygon`

Note: `MakeNGonPoints` returns a list like:

```plaintext
[[x1, x2, x3,...],[y1,y2,y3,...]]
```

while `Polygon()` takes the X and Y lists independently.

Usage: `MakeNGonPoints(<radius>, <num_peaks>)`

Example:

```plaintext
window <- MakeWindow()
ngonp <- MakeNGonPoints(50,10)
ngon <- Polygon(200,200,First(ngonp),Nth(ngonp,2),
                 MakeColor("red"),1)
AddObject(ngon,window)
Draw()
```

See Also: `MakeStarPoints`, `Polygon`, `RotatePoints`, `ZoomPoints`
Name/Symbol: MakePulldown()

Description: Creates a pulldown list that can be used to select an option. The closed version is always 20 pixels high. When opened, it will be by default 15 rows high, although this is made smaller if the pulldown is close to the bottom of the screen. A button is a custom object made from images and text. It has a property 'clickon' that is bound to 'Pulldown'

A closed pulldown will look like this:

![Closed Pulldown]

An open pulldown will look like this:

![Open Pulldown]

Usage: MakePulldown(options, x, y, window, fontsize, width, selected)

The options argument is a list of options you want to appear. x and y are the coordinates of the upper left corner, window is the name of the window (or other graphical object) it appears on, fontsize is the size of the font, and width is the width of the pulldown in pixels. The selected argument is the initial selected list item.

Pulldown objects have a property maxitems, that specify how many elements are displayed. If the list contains more than obj.maxitems, the pulldown will enable scrolling. A pulldown’s click-on handler is by default bound to the 'Pulldown' function. When Pulldown(obj,mousexy) is called, it will pop open the pulldown, allow for a new option to be selected, and return. It returns the index of the selected object, but the selected index can also be accessed using obj.selected.
Chapter 8. Detailed Function and Keyword Reference

Example: See ui.pbl in the demo directory for examples of the use of pulldowns. Pulldowns are also used within the PEBL launcher for various purposes. A basic example is:

```r
options <- MakePulldownButton(["A","B","C"],
                               400,250,gWin,14,100,1)
resp <- WaitForClickOntarget([options],[1])
CallFunction(options.clickon,[options,gClick])
```

See Also: PullDown(), DrawPulldown(), UpdatePulldown

Name/Symbol: MakeScrollBox()

Description: Creates a graphical object that displays and allows selection of a list of items, and scrolls if the text gets too big.

It has a property 'clickon' that is bound to 'ClickOnScrollBox'

A Scrolling textbox looks like this:

![Scrolling Textbox](image)

Usage: MakeScrollBox(list,header,x,y,window,fontsize, width,height,selected)
Chapter 8. Detailed Function and Keyword Reference

The list argument is a text block you want to display. header is a label. x and y are the coordinates of the upper left corner, window is the name of the window (or other graphical object) it appears on, fontsize is the size of the font, and width and height is the size of the scrollbox in pixels. selected indicates which option is selected, and this selection (accessed via .selected) is updated by users using ClickOnScrollBox, which is bound to the .clickon property.

Several related function help update and draw a scrollbox. To change the list or selected item, set the .list property to a new list or .selected to new selection and then call UpdateScrollBox. The function DrawScrollbox to manage redrawing drawing, and ClickOnScrollBox to handle interaction (this is bound to the .clickon property). .inside is bound to InsideTB. A summary of important properties:

- selected: which item is selected
- numitems: How many items on the list
- maxoffset: The most lines that can be displayed
- list: the list of options
- inside: bound to InsideTB
- clickon: bound to ClickOnScrollBox

Example: See ui.pbl in the demo directory for examples of the use of a scrolling text box

```
sb <- MakeScrollBox(Sequence(1,50,1),"The numbers",40,40,gWin,12,150,500,3)
Draw()
resp <- WaitForClickOntarget([sb],[1])
CallFunction(sb.clickon,[sb,gClick])  
#Alternately: ClickOnScrollbox(sb,gClick)
```

See Also: SetScrollingText MakeScrollingTextBox UpdateScrollBox DrawScrollBox ClickOnScrollBox

Name/Symbol: MakeScrollingTextBox()

Description: Creates a graphical object that displays a block of text, and scrolls if the text gets too big. It uses a Scrollbox as its base, but handles parsing the text into lines and hides the selection box. Thus, no 'selection' is displayed (although it actually exists), and a .text property is added to hold the text being displayed.
Chapter 8. Detailed Function and Keyword Reference

It has a property 'clickon' that is bound to 'ClickOnScrollBox'

A Scrolling textbox looks like this:

```
define Start(p)
    gVideoWidth<-800
gVideoHeight<-600
gWin <- MakeWindow("grey90")
gSleepEasy <- 1
```

Usage:

```
MakeScrollingTextBox(text,x,y,window,fontsize,
width,height,linewrap)
```

The text argument is a text block you want to display. x and y are the coordinates of the upper left corner, window is the name of the window (or other graphical object) it appears on, fontsize is the size of the font, and width and height is the size of the scrolling textbox in pixels. linewrap, if non-zero, will parse the text layout so you see everything, breaking when the text gets to the end of the box, and on linebreaks. if 0, it will only break at explicit carriage returns.

Note that parsing text into the scrolling textbox is fairly fast, but you may need workarounds for to display extremely long files if you want high responsiveness.

Several related functions help update and draw a scrolling textbox. To change the text, use SetScrollingText. Because a scrolling textbox is really just a scrollbox, you also use DrawScrollBox to manage drawing, and ClickOnScrollBox to handle interaction (this is bound to the .clickon property). .inside is bound to InsideTB

Example: See ui.pbl in the demo directory for examples of the use of a scrolling text box

```
textscroll <- MakeScrollingTextBox("",200,50,gWin,12,
300,150,0)

SetScrollingText(textscroll,FileReadText("Uppercase.txt"))
Draw()
resp <- WaitForClickOntarget([textscroll],[1])
CallFunction(textscroll.clickon,[textscroll,gClick])
```
Name/Symbol: **MakeSineWave()**

Description: Creates a sine wave that can be played using the Play() or PlayBackground() functions. It will create a single-channel sound at 44100 bitrate, 16 bit precision.

Usage: `MakeSineWave(<duration_in_ms>, <hz>, <amplitude>)`

- The first argument specifies how long (in ms) the tone should be.
- The second argument specifies the frequency. Good values range between 100 and 2000.
- The third argument specifies the volume. It should be less than 1.0.

Example:
```
##Make a sound that is 1000 ms, but just play 300 ms
sound <- MakeSineWave(200, 220, 1000)
PlayBackground(sound)
Wait(300)
Stop(sound)
```

See Also: `PlayForeground()`, `PlayBackground()`, `Stop()`

Name/Symbol: **MakeStarPoints()**

Description: Creates a set of points that form a regular star. It can be transformed with functions like RotatePoints, or it can be used to create a graphical object with Polygon.

Note: `MakeStarPoints` returns a list:
```
[[x1, x2, x3,...],[y1,y2,y3,...]],
```

while Polygon() takes the X and Y lists independently.

Usage: `MakeStarPoints(<outer_radius>, <inner_radius>, <num_peaks>)`
Chapter 8. Detailed Function and Keyword Reference

Example:

```r
window <- MakeWindow()
sp <- MakeStarPoints(50,20,10)
star <- Polygon(200,200,First(sp),Nth(sp,2),
                MakeColor("red"),1)
AddObject(star,window)
Draw()
```

See Also: MakeNGonPoints, Polygon, RotatePoints, ZoomPoints

Name/Symbol: MakeTextBox()

Description: Creates a textbox in which to display text. Textboxes allow multiple lines of text to be rendered; automatically breaking the text into lines.

Usage: `MakeWindow(<text>,<font>,<width>,<height>)`

Example:

```r
font <-MakeFont("Vera.ttf", 1, 12, MakeColor("red"), MakeColor("green"), 1)
tb <- MakeTextBox("This is the text in the textbox", font, 100, 250)
```

See Also: MakeLabel(), GetText(), SetText(), SetCursorPosition(), GetCursorPosition(), SetEditable()

Name/Symbol: MakeWindow()

Description: Creates a window to display things in. Background is specified by `<color>`.

Usage: `MakeWindow(<color>)`

Example:

```r
win <- MakeWindow()
gWin <- MakeWindow("white")
```

See Also:

Name/Symbol: MakeTextList()

Description: This takes a list and creates a block of text with carriage returns, ensuring each item of the list is on its own line; it also requires an offset, skipping the first lines of the list. It is mostly a helper function used by Scrollbox objects to help format. It will make text out of the entire list, so you should be sure to cut off the end for efficiency if you only want to display some of the lines.
Usage: \texttt{MakeTextList([<list>], <list-offset>, <prebuffer>)}

Example: 
\begin{verbatim}
letters <- FileReadList("Uppercase.txt")
out <- MakeTextList(letters,20,"--")
The above code will create the following:

--u
--v
--w
--x
--y
--z
\end{verbatim}

See Also: \texttt{ListToString}

Name/Symbol: \texttt{Match()}

Description: Returns a list of 0/1, indicating which elements of \texttt{<list>} match \texttt{<target>}

Usage: \texttt{Match(<list>,target)}

Example: 
\begin{verbatim}
x <- [1,2,3,3,2,2,1]
Print(Match(x,1)) \#== [1,0,0,0,0,0,1]
Print(Match(x,2)) \#== [0,1,0,0,1,1,0]
Print(Match(x,3)) \#== [0,0,1,1,0,0,0]
\end{verbatim}

See Also: \texttt{Filter()}, \texttt{Subset()}, \texttt{Lookup()}

Name/Symbol: \texttt{Max()}

Description: Returns the largest of \texttt{<list>}

Usage: \texttt{Max(<list>)}

Example: 
\begin{verbatim}
c <- [3,4,5,6]
m <- Max(c) \# m == 6
\end{verbatim}

See Also: \texttt{Min()}, \texttt{Mean()}, \texttt{StDev()}

Name/Symbol: \texttt{MD5Sum()}

Description: Computes MD5 sum of a text string. Returns blank if no string provided\texttt{<list>}

184
Chapter 8. Detailed Function and Keyword Reference

Usage: MD5Sum(<text>)

Example:

```
Print(MD5Sum(""))
#Return: d41d8cd98f00b204e9800998ecf8427e
Print(MD5Sum("bananana"))
#returns bb8e9af523e4aeffe88f1807fb2af9ce

text <- FileReadText("test.pbl")
Print(MD5Sum(text))
#returns: 3396a651bd3c96f9799ce02eebc48801; see similar example next
Print(MD5File("test.pbl"))
# returns 3396a651bd3c96f9799ce02eebc48801

Print(MD5File("doesnotexist.txt"))
#returns 0
```

See Also: MD5File()

Name/Symbol: MD5File()

Description: Computes MD5 sum of a file. Returns blank if no string provided.<list>.

Usage: MD5File(<filename>)

Example:

```
text <- FileReadText("test.pbl")
Print(MD5Sum(text))
#returns: 3396a651bd3c96f9799ce02eebc48801; see similar example next

Print(MD5File("test.pbl"))
# returns 3396a651bd3c96f9799ce02eebc48801

Print(MD5File("doesnotexist.txt"))
#returns 0

Print(MD5Sum(""))
#Return: d41d8cd98f00b204e9800998ecf8427e
Print(MD5Sum("bananana"))
#returns bb8e9af523e4aeffe88f1807fb2af9ce
```

See Also: MD5Sum()
Name/Symbol: \texttt{Mean()}
Description: Returns the mean of the numbers in \texttt{<list>}.  
Usage: \texttt{Mean(<list-of-numbers>)}
Example:  
\begin{verbatim}
c <- [3,4,5,6] 
m <- Mean(c) \# m == 4.5
\end{verbatim}
See Also: \texttt{Median()}, \texttt{Quantile()}, \texttt{StDev()}, \texttt{Min()}, \texttt{Max()}

Name/Symbol: \texttt{Median()}
Description: Returns the median of the numbers in \texttt{<list>}.  
Usage: \texttt{Median(<list-of-numbers>)}
Example:  
\begin{verbatim}
c <- [3,4,5,6,7] 
m <- Median(c) \# m == 5
\end{verbatim}
See Also: \texttt{Mean()}, \texttt{Quantile()}, \texttt{StDev()}, \texttt{Min()}, \texttt{Max()}

Name/Symbol: \texttt{Merge()}
Description: Combines two lists, \texttt{<lista>} and \texttt{<listb>}, into a single list.  
Usage: \texttt{Merge(<lista>,<listb>)}
Example:  
\begin{verbatim}
Merge([1,2,3],[8,9]) \# == [1,2,3,8,9]
\end{verbatim}
See Also: \texttt{[ ]}, \texttt{Append()}, \texttt{List()}

Name/Symbol: \texttt{MessageBox()}
Description: Hides what is on the screen and presents a textbox with specified message, with a button to click at the bottom to continue.  
Usage: \texttt{MessageBox(<message>,<window>)}
Example:  
\begin{verbatim}
gWin <- MakeWindow() 
MessageBox("Click below to begin.",gWin)
\end{verbatim}
See Also: \texttt{GetEasyInput}, \texttt{EasyTextBox}
Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: Min()
Description: Returns the ‘smallest’ element of a list.
Usage: Min(<list>)
Example: c <- [3, 4, 5, 6]
        m <- Min(c) # == 3
See Also: Max()

Name/Symbol: Mod()
Description: Returns <num>, <mod>, or remainder of <num>/<mod>
Usage: Mod(<num> <mod>)
Example: Mod(34, 10) # == 4
        Mod(3, 10) # == 3
See Also: Div()

Name/Symbol: Move()
Description: Moves an object to a specified location. Images and Labels are moved according to their center; TextBoxes are moved according to their upper left corner.
Usage: Move(<object>, <x>, <y>)
Example: Move(label, 33, 100)
See Also: MoveCorner(), MoveCenter(), .X and .Y properties.

Name/Symbol: MoveCenter()
Description: Moves a TextBox to a specified location according to its center, instead of its upper left corner.
Usage: MoveCenter(<object>, <x>, <y>)
Example: MoveCenter(TextBox, 33, 100)
See Also: Move(), MoveCenter(), .X and .Y properties
Name/Symbol: MoveCorner()

Description: Moves a label or image to a specified location according to its upper left corner, instead of its center.

Usage: MoveCorner(<object>, <x>, <y>)

Example: MoveCorner(label, 33, 100)

See Also: Move(), MoveCenter(), .X and .Y properties

Name/Symbol: MoveObject()

Description: Calls the function named by the .move property of a custom object. Useful if a custom object has complex parts that need to be moved; you can bind .move to a custom move function and then call it (and anything else) using MoveObject. MoveObject will fall back on a normal move, so you can handle movement of many built-in objects with it.

Usage: MoveObject(obj,x,y)

Example:

```plaintext
##This overrides buttons placement at the center:
done <- MakeButton("QUIT",400,250,gWin,150)
done.move <- "MoveCorner"
MoveObject(done, 100,100)
```

See Also: Inside(), Move ClickOn, DrawObject
8.14 Name/Symbol: NonOverlapLayout

Description: Creates a set of num points in a xy range, that have a (soft) minimum tolerance of \(<\text{tol}\) between points. That is, to the extent possible, the returned points will have a minimum distance between them of \(<\text{tol}\>\). This may not be possible or be very difficult, and so after a limited number of attempts (by default, 100), the algorithm will return the current configuration, which may have some violations of the minimum tolerance rule, but it will usually be fairly good.

The algorithm works by initializing with a random set of points, then computing a pairwise distance matrix between all points, finding the closest two points, and resampling one of them until its minimum distance is larger than the current. Thus, each internal iteration uniformly improves (or keeps the configuration the same), and the worst points are reconfigured first, so that even if a configuration that does not satisfy the constraints, it will usually be very close.

Internally, the function (located in pebl-lib/Graphics.pbl) has a variable that controls how many steps are taken, called “limit”, which is set to 100. For very compacted or very large iterations, this limit can be increased by editing the file or making a copy of the function.

The function usually returns fairly quickly, so it can often be used real-time between trials. However, for complex enough configurations, it can take on the order of seconds; furthermore, more complex configurations might take longer than less complex configurations, which could represent a potential confound (if more complex stimuli have longer ISIs). Users should thus consider creating the configurations when the test is initialized, or created prior to the study and then saved out to a file for later use.
Chapter 8. Detailed Function and Keyword Reference

Usage:

NonOverlapLayout(<xmin>,<xmax>,<ymin>,<ymax>,<tol>,<num>)

Example:
Example PEBL Program using NonoverlapLayout:

define Start(p)
{
   win <- MakeWindow()
   ## Make 25 points in a square in the middle
   ## of the screen, a minimum of 50 pixels apart.
   ## This is too compact, but it will be OK.
   points <- NonOverlapLayout(100,300,200,400,50,25)
   circs <- []
   ## This should non-overlapping circles of radius 25
   loop(i,points)
   {
      tmp <- Circle(First(i),Second(i),25,
                     MakeColor("blue"),0)
      AddObject(tmp,win)
      circs <- Append(circs,tmp)
   }

   rect1 <- Square(200,300,200,MakeColor("black"),0)
   rect2 <- Square(600,300,200,MakeColor("black"),0)
   AddObject(rect1,win)
   AddObject(rect2,win)
   ## Reduce the tolerance: this one should be better
   points <- NonOverlapLayout(500,700,200,400,50,15)
   ## This should non-overlapping circles of radius 15
   loop(i,points)
   {
      tmp <- Circle(First(i),Second(i),15,
                     MakeColor("blue"),0)
      AddObject(tmp,win)
      circs <- Append(circs,tmp)
   }
   Draw()
   WaitForAnyKeyPress()
}

190
The output of the above program is shown below. Even for the left configuration, which is too compact (and which takes a couple seconds to run), the targets are fairly well distributed.

See Also: LayoutGrid()

---

Name/Symbol: not
Description: Logical not
Usage:
Example:
See Also: and, or

---

Name/Symbol: NormalDensity()
Description: Computes density of normal standard distribution
Usage: NormalDensity(<x>)
Example:

```
Print(NormalDensity(-100))     # 1.8391e-2171
Print(NormalDensity(-2.32635)) # 5.97
Print(NormalDensity(0))        # 0.398942
```
Chapter 8. Detailed Function and Keyword Reference

Print(NormalDensity(1.28155)) # .90687
Print(NormalDensity(1000))   # inf

See Also: RandomNormal(), CumNormInv()

Name/Symbol: Nth()

Description: Extracts the Nth item from a list. Indexes from 1 upwards. Last() provides faster access than Nth() to the end of a list, which must walk along the list to the desired position.

Usage: Nth(<list>, <index>)

Example:

a <- ["a", "b", "c", "d"]
Print(Nth(a,3))  # == 'c'
See Also: First(), Last()

Name/Symbol: NthRoot()

Description: <num> to the power of 1/<root>.

Usage: NthRoot(<num>, <root>)

Example:

See Also:
Chapter 8. Detailed Function and Keyword Reference

8.15 O

Name/Symbol: OpenCOMPort
Description: This opens a COM/Serial port
Usage: OpenCOMPort(<portnum>,<baud>)
Example: 
See Also: COMPortGetByte, COMPortSendByte, OpenPPort, SetPPortMode, GetPPortMode

Name/Symbol: OpenJoystick
Description: This opens an available joystick, as specified by its index. The returned object can then be used to access the state of the joystick. It takes an integer argument, and for the most part, if you have a single joystick attached to your system, you will use OpenJoystick(1). If you want to use a second joystick, use OpenJoystick(2), and so on.
Usage: OpenJoystick()
Example: See joysticktest.pbl in the demo directory
See Also: GetNumJoysticks(), OpenJoystick(), GetNumJoystickAxes() GetNumJoystickBalls(), GetNumJoystickButtons(), GetNumJoystickHats() GetJoystickAxisState(), GetJoystickHatState(), GetJoystickButtonType()

Name/Symbol: OpenNetworkListener()
Description: Creates a network object that listens on a particular port, and is able to accept incoming connections. You can use CheckForNetworkConnections to accept incoming connections. This is an alternative to the WaitForNetworkConnection function that allows more flexibility (and allows updating the during waiting for the connection).
Usage: net <- OpenNetworkListener(port)
Example:

```r
network <- OpenNetworkListener(4444)
time <- GetTime()
while(not connected and (GetTime() < time + 5000)) {
    connected <- CheckForNetworkConnection(network)
}
```

See Also: CheckForNetworkConnection(), Getdata(), WaitForNetworkConnection(), CloseNetwork()

---

**Name/Symbol:** `OpenSubMenus()`

**Description:** Used by `ClickOnMenu` to open, display a submenu and get a click.

**Usage:**

```r
OpenSubMenus(obj, [x, y])
```

This function is bound to the `.clickon` property of a menu. It will open and display all the submenus, wait for a click, and execute the function called.

**Example:** This creates a menu and awaits clicking on. More complete examples are available in ui.pbl. It requires that `MyMessage` is created somewhere.

```r
menu1 <- MakeMenuItem("File", 0, 0, gWin, 14, 10, "MYMESSAGE")

menu2 <- MakeMenu("Edit", 70, 0, gWin, 14, 10, "MYMESSAGE")

menus <- [menu1, menu2]
opt <- WaitForClickOnTarget(menu, [1, 2])
ClickOnMenu(Nth(menus, opt), gClick)
```

See Also: MakeMenu(), OpenSubMenus(), MakeMenuItem

---

**Name/Symbol:** `or`

**Description:** Logical or

**Usage:**

**Example:**
Name/Symbol: `OpenPPort`
Description: Opens a Parallel port, returning an object that can be used for parallel port communications.
Usage: `OpenPPort(<name>)` The `<name>` argument can be one of "LPT1", "LPT2", and "LPTX". Most likely, a parallel port will be configured to LPT1, but other configurations are sometimes possible.
Example: See Also: `COMPortGetByte`, `COMPortSendByte`, `OpenCOMPort`, `SetPPortMode`, `GetPPortMode`

Name/Symbol: `Order()`
Description: Returns a list of indices describing the order of values by position, from min to max.
Usage: `Order(<list-of-numbers>)`
Example: `n <- [33,12,1,5,9]
  o <- Order(n)
  Print(o) #should print [3,4,5,2,1]
See Also: `Rank()`
Chapter 8. Detailed Function and Keyword Reference

8.16 P

Name/Symbol: PausePlayback()

Description: Pauses a playing movie or audio stream. This is used for movies whose playback was initiated using StartPlayback, which then ran as background threads during a Wait() function.

Usage: PausePlayback(movie)

Example:
```
movie <- LoadMovie("movie.avi", gWin, 640, 480)
PrintProperties(movie)
Move(movie, 20, 20)
Draw()
StartPlayback(movie)
Wait(500) #Play 500 ms of the movie.
PausePlayback(movie)
Wait(500)
```

See Also: LoadAudioFile(), LoadMovie(), PlayMovie(), StartPlayback()

Name/Symbol: PlayForeground()

Description: Plays the sound 'in the foreground'; does not return until the sound is complete.

Usage: PlayForeground(<sound>)

Example:
```
sound <- MakeSineWave(200, 220, 1000)
PlayForeground(sound)
```

See Also: PlayBackground(), Stop()

Name/Symbol: PlayBackground()

Description: Plays the sound 'in the background', returning immediately.

Usage: PlayBackground(<sound>)

Example:
```
sound <- MakeSineWave(200, 220, 1000)
PlayBackground(sound)
Wait(300)
Stop(sound)
```
Chapter 8. Detailed Function and Keyword Reference

See Also: \texttt{PlayForeground()}, \texttt{Stop()}

---

Name/Symbol: \texttt{PlayMovie()}

Description: Plays the movie (or other multimedia file) loaded via either the \texttt{LoadMovie} or \texttt{LoadAudioFile} function. Note that this functionality uses a different underlying system than the sound playing functions \texttt{PlayBackground} and \texttt{PlayForeground}, and they are not interchangeable.

Usage: \texttt{PlayMovie(movie)}

Example:

\begin{verbatim}
movie <- LoadMovie("movie.avi",gWin,640,480)
PrintProperties(movie)
Move(movie,20,20)
movie.volume <- .1
status <- EasyLabel("Demo Movie Player",300,25,gWin,22)
Draw()
PlayMovie(movie)
\end{verbatim}

See Also: \texttt{LoadAudioFile()}, \texttt{LoadMovie()}, \texttt{StartPlayback()}, \texttt{PausePlayback()}

---

Name/Symbol: \texttt{Plus}

Description: Creates a polygon in the shape of a plus sign. Arguments include position in window.

- \(<x>\) and \(<y>\) is the position of the center
- \(<\text{size}>\) or the size of the plus sign in pixels
- \(<\text{width}>\) thickness of the plus
- \(<\text{color}>\) is a color object (not just the name)

Like other drawn objects, the plus must then be added to the window to appear.

Usage: \texttt{Plus(x,y,size,width,color)}

Example:

\begin{verbatim}
win <- MakeWindow()
p1 <- Plus(100,100,80,15,MakeColor("red"))
AddObject(p1,win)
Draw()
\end{verbatim}

See Also: \texttt{BlockE()}, \texttt{Polygon()}, \texttt{MakeStarPoints()}, \texttt{MakeNGonPoints()}

197
Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: Polygon

Description: Creates a polygon in the shape of the points specified by <xpoints>, <ypoints>. The lists <xpoints> and <ypoints> are adjusted by <x> and <y>, so they should be relative to 0, not the location you want the points to be at. Like other drawn objects, the polygon must then be added to the window to appear.

Usage: 
```
Polygon(<x>, <y>, <xpoints>, <ypoints>,
        <color>, <filled>)
```

Example:
```
win <- MakeWindow()
#This makes a T
xpoints <- [-10,10,10,20,20,-20,-20,-10]
ypoints <- [-20,-20,40,40,50,50,40,40]
p1 <- Polygon(100,100, xpoints, ypoints, 
               MakeColor("black"), 1)
AddObject(p1, win)
Draw()
```

See Also: BlockE(), Bezier(), MakeStarPoints(), MakeNGonPoints()

Name/Symbol: PopUpEntryBox()

Description: Creates a small text-entry box at a specified location.

Usage: 
```
PopUpEntryBox(<text>, <win>, [x,y])
```

Example:
```
subnum <- PopUpEntryBox("Enter participant code", gWin, [100,100])
```

See Also: MessageBox GetEasyInput, PopUpMessageBox

Name/Symbol: PopUpMessageBox()

Description: Creates a small 300x200 information box at the current cursor location, but also adjusts so it is on the screen. It must be dismissed by clicking the 'OK' button.

Usage: 
```
PopUpMessageBox(<text>, <win>)
```

Note that the function puts the box on the screen at the current mouse position. If you want control over where it goes, you need to use SetMouseCursorPosition immediately before the box is made.
Chapter 8. Detailed Function and Keyword Reference

Example:

```
subnum <- PopUpMessageBox("There has been an error.", gWin)
```

See Also:

```
MessageBox GetEasyInput, PopUpEntryBox
```

Name/Symbol: **Print()**

Description: Prints `<value>` to stdout (the console [Linux] or the file `stdout.txt` [Windows]), and then appends a newline afterwards.

Usage:

```
Print(<value>)
```

Example:

```
Print("hello world")
Print(33 + 43)
x <-Print("Once")
```

See Also:

```
Print_(), FilePrint()
```

Name/Symbol: **Pow()**

Description: Raises or lowers `<num>` to the power of `<pow>`.  

Usage:

```
Pow(<num>, <pow>)
```

Example:

```
Pow(2, 6) # == 64
Pow(5, 0) # == 1
```

See Also:

```
```

Name/Symbol: **Print()**

Description: Prints `<value>` to stdout (the console [Linux] or the file `stdout.txt` [Windows]), and then appends a newline afterwards.

Usage:

```
Print(<value>)
```

Example:

```
Print("hello world")
Print(33 + 43)
x <-Print("Once")
```

See Also:

```
Print_(), FilePrint()
```

199
Name/Symbol: PrintProperties()

Description: Prints properties/values for any complex object. These include textboxes, fonts, colors, images, shapes, etc. Mostly useful as a debugging tool.

Usage: PrintProperties(<object>)

Example:

```
win <- MakeWindow()
tb <- EasyTextbox("one",20,20,win,22,400,80)
PrintProperties(tb)
```

##Output:

```
----------
[CURSORPOS]: 0
[EDITABLE]: 0
[HEIGHT]: 80
[ROTATION]: 0
[TEXT]: one
[VISIBLE]: 1
[WIDTH]: 400
[X]: 20
[Y]: 20
[ZOOMX]: 1
[ZOOMY]: 1
----------
```

See Also: Print()

Name/Symbol: Print_

Description: Prints <value> to stdout; doesn’t append a newline afterwards.

Usage: Print_(<value>)

Example:

```
Print_("This line")
Print_(" ")
Print_("and")
Print_(" ")
Print("Another line")
# prints out: 'This line and Another line'
```

200
See Also:  Print(), FilePrint()

Name/Symbol: PrintList()

Description:  Prints a list, without the ’,’s or ‘|’ characters. Puts a carriage return at the end. Returns a string that was printed. If a list contains other lists, the printing will wrap multiple lines and the internal lists will be printed as normal. To avoid this, try PrintList(Flatten(list)).

Usage:  PrintList( <list>)

Example:  PrintList( [1,2,3,4,5,5,5])

##
## Produces:
## 1 2 3 4 5 5 5
PrintList([[1,2],[3,4],[5,6]])
# Produces:
# [1,2]
#, [3,4]
#, [5,6]

PrintList(Flatten([[1,2],[3,4],[5,6]]))
# Produces:
# 1 2 3 4 5 6

See Also:  Print(), Print_(), FilePrint(), FilePrint_(), FilePrintList(),

Name/Symbol: PropertyExists

Description:  Tests whether a particular named property exists. This works for custom or built-in objects. This is important to check properties that might not exist, because trying to GetProperty of a non-existent property will cause a fatal error.

Usage:  out <- PropertyExists(obj,property)

Example:  obj <- MakeCustomObject("myobject")
obj.taste <- "buttery"
obj.texture <- "creamy"
SetProperty(obj,"flavor","tasty")

list <- GetPropertyList(obj)


Chapter 8. Detailed Function and Keyword Reference


code
loop(i,list)
{
    if(PropertyExists(obj,i))
    {
        Print(i + " : " + GetProperty(obj,i))
    }
}

See Also: GetPropertyList, GetProperty, SetProperty
MakeCustomObject, PrintProperties

Name/Symbol: Pulldown()

Description: This handles making a new selection on a pulldown box.

Usage: Pulldown(object, [x,y])

This function is typically the primary way of interacting with a pulldown box. It will have the effect of opening the pull-down box, waiting for the user to select a new option, and then changing the selected option to whatever they click on.

Example: See demo ui.pbl for examples of the use of pulldowns. Pulldowns are also used within the PEBL launcher for various purposes. A basic example is:

options <- MakePulldownButton(\"A\",\"B\",\"C\"),400,250,gWin,14,100,1)
resp <- WaitForClickOntarget([options],[1])
newvalue <- Pulldown(options,gClick)

See Also: MakePullDown(), DrawPulldown(), UpdatePulldown

Name/Symbol: PushButton

Description: Animates a button-pushing. It takes a button created using the MakeButton function and will animate a downclick when the mouse is down, and release when the mouse is unclicked. To conform with general object handlers, it requires specifying a mouse click position, which could be [0,0], or gclick. This function is bound to the property 'clickon' of any button, allowing you to handle mouse clicks universally for many different objects.

202
Chapter 8. Detailed Function and Keyword Reference

Usage: PushButton(button, xylist)

Example: The following creates a button, waits for you to click on it, and animates a button press

```plaintext
done <- MakeButton("QUIT",400,250,gWin,150)
resp <- WaitForClickOnTarget([done],[1])
PushButton(done,[0,0])
```

To handle multiple buttons, you can do:

```plaintext
done <- MakeButton("QUIT",400,250,gWin,150)
ok <- MakeButton("OK",400,250,gWin,150)
resp <- 2
while (resp != 1)
{
    Draw()
    resp <- WaitForClickOnTarget([done,ok],[1,2])
    obj <- Nth([done,ok],resp)
    CallFunction(obj.clickon,[obj,gClick])
}
```

See Also: MakeCheckBox()

Name/Symbol: PushOnEnd

Description: Pushes an item onto the end of a list, modifying the list itself.

Note: PushOnEnd is a more efficient replacement for Append(). Unlike Append, it will modify the original list as a side effect, so the following works:

```
PushOnEnd(list, item)
```

There is no need to set the original list to the result of PushOnEnd, like you must do with Append. However, it does in fact work, and incurs only a slight overhead, so that Append can often be replaced with PushOnEnd without worry.

```
list <- PushOnEnd(list, item)
```

Usage: PushOnEnd(<list>, <item>)

203
Example:

```lisp
list <- Sequence(1,5,1)
double <- []
loop(i, list)
{
    PushOnEnd(double, [i,i])
}
Print(double)
# Produces [[1,1],[2,2],[3,3],[4,4],[5,5]]
```

See Also: SetElement(), List(), [], Merge(), PushOnEnd
8.17 Q

Name/Symbol: Quantile()

Description: Returns the <num> quantile of the numbers in <list>. <num> should be between 0 and 100

Usage: Quantile(<list>, <num>)

Example: 
```
##Find 75th percentile to use as a threshold.
thresh <- Quantile(rts, 75)
```

See Also: StDev(), Median(), Mean(), Max(), Min()
8.18 R

Name/Symbol: RadToDeg()
Description: Converts \(<rad>\) radians to degrees.
Usage: RadToDeg( <rad>)
Example: 
See Also: DegToRad(), Tan(), Cos(), Sin(), ATan(), ASin(), ACos()

Name/Symbol: Random()
Description: Returns a random number between 0 and 1.
Usage: Random()
Example: a <- Random()
See Also: Random(), RandomBernoulli(), RandomBinomial(), RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomUniform(), RandomizeTimer(), SeedRNG()

Name/Symbol: RandomBernoulli()
Description: Returns 0 with probability \((1-<p>)\) and 1 with probability \(<p>\).
Usage: RandomBernoulli(<p>)
Example: RandomBernoulli(.3)
See Also: Random(), RandomBernoulli(), RandomBinomial, RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomUniform(), RandomizeTimer(), SeedRNG()

Name/Symbol: RandomBinomial
Description: Returns a random number according to the Binomial distribution with probability \(<p>\) and repetitions \(<n>\), i.e., the number of \(<p>\) Bernoulli trials that succeed out of \(<n>\) attempts.
Usage: RandomBinomial(<p> <n>)

206
Example: RandomBinomial(.3, 10) # returns number from 0 to 10

See Also: Random(), RandomBernoulli(), RandomBinomial, RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomUniform(), RandomizeTimer(), SeedRNG()

Name/Symbol: RandomDiscrete()
Description: Returns a random integer between 1 and the argument (inclusive), each with equal probability. If the argument is a floating-point value, it will be truncated down; if it is less than 1, it will return 1, and possibly a warning message.
Usage: RandomDiscrete(<num>)
Example: # Returns a random integer between 1 and 30: RandomDiscrete(30)
See Also: Random(), RandomBernoulli(), RandomBinomial, RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomUniform(), RandomizeTimer(), SeedRNG()

Name/Symbol: RandomExponential()
Description: Returns a random number according to exponential distribution with mean <mean> (or decay 1/mean).
Usage: RandomExponential(<mean>)
Example: RandomExponential(100)
See Also: Random(), RandomBernoulli(), RandomBinomial, RandomDiscrete(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomUniform(), RandomizeTimer(), SeedRNG()

Name/Symbol: RandomizeTimer()
Description: Seeds the RNG with the current time.
Usage: RandomizeTimer()
Example: RandomizeTimer()
207
Chapter 8. Detailed Function and Keyword Reference

See Also: Random(), RandomBernoulli(), RandomBinomial(), RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomUniform(), SeedRNG()

Name/Symbol: RandomLogistic()
Description: Returns a random number according to the logistic distribution with parameter \( p \): \[ f(x) = \frac{\exp(x)}{1+\exp(x)} \]
Usage: RandomLogistic(<p>)
Example: RandomLogistic(.3)
See Also: Random(), RandomBernoulli(), RandomBinomial(), RandomDiscrete(), RandomExponential(), RandomLogNormal(), RandomNormal(), RandomUniform(), RandomizeTimer, SeedRNG()

Name/Symbol: RandomLogNormal()
Description: Returns a random number according to the log-normal distribution with parameters \( \text{median} \) and \( \text{spread} \). Generated by calculating \( \text{median} \times \exp(\text{spread} \times \text{RandomNormal}(0, 1)) \). \( \text{spread} \) is a shape parameter, and only affects the variance as a function of the median; similar to the coefficient of variation. A value near 0 is a sharp distribution (1-3), larger values are more spread out; values greater than 2 make little difference in the shape.
Usage: RandomLogNormal(<median>, <spread>)
Example: RandomLogNormal(5000, .1)
See Also: Random(), RandomBernoulli(), RandomBinomial(), RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomNormal(), RandomUniform(), RandomizeTimer, SeedRNG()

Name/Symbol: RandomNormal()
Description: Returns a random number according to the standard normal distribution with \( \text{mean} \) and \( \text{stdev} \).
Usage: RandomNormal(<mean>, <stdev>)

208
**Chapter 8. Detailed Function and Keyword Reference**

Example:

**See Also:** Random(), RandomBernoulli(), RandomBinomial, RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomUniform(), RandomizeTimer, SeedRNG()

---

**Name/Symbol:** RandomUniform()

**Description:** Returns a random floating-point number between 0 and <num>.

**Usage:** RandomUniform(<num>)

**Example:**

See Also: Random(), RandomBernoulli(), RandomBinomial, RandomDiscrete(), RandomExponential(), RandomLogistic(), RandomLogNormal(), RandomNormal(), RandomizeTimer(), SeedRNG()

---

**Name/Symbol:** Rank()

**Description:** Returns a list of numbers describing the rank of each position, from min to max. The same as calling Order(Order(x)).

**Usage:** Rank(<list-of-numbers>)

**Example:**

```
n <- [33,12,1,5,9]
o <- Rank(n)
Print(o) #should print [5,4,1,2,3]
```

**See Also:** Order()

---

**Name/Symbol:** ReadCSV()

**Description:** Reads a comma-separated value file into a nested list. Need not be named with a .csv extension. It should properly strip quotes from cells, and not break entries on commas embedded within quoted text.

**Usage:** ReadCSV(<filename>)

**Example:**

```
table <- ReadCSV("datafile.csv")
```

**See Also:** FileReadTable(), FileReadList, StripQuotes

209
Name/Symbol: Rectangle()

Description: Creates a rectangle for graphing at x,y with size dx and dy. Rectangles are only currently definable oriented in horizontal/vertical directions. A rectangle must be added to a parent widget before it can be drawn; it may be added to widgets other than a base window. The properties of rectangles may be changed by accessing their properties directly, including the FILLED property which makes the object an outline versus a filled shape.

Usage: Rectangle(<x>, <y>, <dx>, <dy>, <color>)

Example:

```r <- Rectangle(30,30,20,10, MakeColor(green))
AddObject(r, win)
Draw()```

See Also: Circle(), Ellipse(), Square(), Line()

---

Name/Symbol: ReflectPoints

Description: Takes a set of points (defined in a joined list [[x1,x2,x3,...],[y1,y2,y3,...]] and reflects them around the vertical axis x=0, returning a similar [[x],[y]] list. Identical to ZoomPoints(pts,-1,1)

Usage: ReflectPoints(<points>)

Example:

```points <- [[1,2,3,4],[20,21,22,23]]
newpoints <- ReflectPoints(points)```

See Also: ZoomPoints(), RotatePoints

---

Name/Symbol: RegisterEvent()

Description: Adds an event to the event loop. This function is currently experimental, and its usage may change in future versions of PEBL.

Usage: USAGE CURRENTLY UNDOCUMENTED

Example:
Chapter 8. Detailed Function and Keyword Reference

See Also: ClearEventLoop(), StartEventLoop()

Name/Symbol: RemoveFile()
Description: Removes a file from the file system.
Usage: RemoveObject(<filename>)
Example:
```
tmpfile <- FileOpenWrite("tmp.txt")
FilePrint(tmpfile,Random())
FileClose(tmpfile)
text <- FileReadText("tmp.txt")
RemoveFile("tmp.txt")
```
See Also: GetDirectoryListing(), FileExists(), IsDirectory(), MakeDirectory()

Name/Symbol: RemoveObject()
Description: Removes a child widget from a parent. Useful if you are adding a local widget to a global window inside a loop. If you do not remove the object and only Hide() it, drawing will be sluggish. Objects that are local to a function are removed automatically when the function terminates, so you do not need to call RemoveObject() on them at the end of a function.
Usage: RemoveObject(<object>, <parent>)
Example:
See Also:

Name/Symbol: RemoveSubset()
Description: Removes a subset of elements from a list. Creates a new list, and does not affect the original
Usage: RemoveSubset(<list1>,<list-of-element-indices>)
Example:
```
list1 <- [1,2,2,4,5]
list2 <- RemoveSubset(list1,[2,3])
Print(list1) #[1,2,2,4,5]
Print(list2) #[1,4,5]
```

211
See Also: Merge(), Insert(), Rest()

**Name/Symbol:** Repeat()

**Description:** Makes and returns a list by repeating `<object> <n>` times. Has no effect on the object. Repeat will not make new copies of the object. If you later change the object, you will change every object in the list.

**Usage:** `Repeat(<object>, <n>)`

**Example:**
```r
x <- "potato"
y <- repeat(x, 10)
Print(y)
# produces ["potato","potato","potato",
  "potato","potato", "potato",
  "potato","potato","potato","potato"]
```

See Also: RepeatList()

---

**Name/Symbol:** RepeatList()

**Description:** Makes a longer list by repeating a shorter list `<n>` times. Has no effect on the list itself, but changes made to objects in the new list will also affect the old list.

**Usage:** `RepeatList(<list>, <n>)`

**Example:**
```r
RepeatList([1,2],3) # == [1,2,1,2,1,2]
```

See Also: Repeat(), Merge(), [ ]

---

**Name/Symbol:** Replace()

**Description:** Creates a copy of a (possibly nested) list in which items matching some list are replaced for other items. `<template>` can be any data structure, and can be nested. `<replacementList>` is a list containing two-item list pairs: the to-be-replaced item and to what it should be transformed.

Note: replacement searches the entire `<replacementList>` for matches. If multiple keys are identical, the item will be replaced with the last item that matches.

**Usage:** `Replace(<template>,<replacementList>)`
Example:

```r
x <- c("a", "b", "c", "x")
rep <- list(c("a", "A"), c("b", "B"), c("x", "D"))
Print(Replace(x, rep))
# Result: [A, B, c, D]
```

See Also: `ReplaceChar()`

---

Name/Symbol: `ReplaceChar()`

Description: Substitutes `<char2>` for `<char>` in `<string>`. Useful for saving subject entry data in a file; replacing spaces with some other character. The second argument can either be a character to match, or a list of characters to match, in which case they all get replaced with the third argument.

Usage:

- `ReplaceChar(<string>, <char>, <char2>)`
- `ReplaceChar(<string>, list(<chara>, <charb>), <char2>)`

Example:

```r
x <- c("Sing a song of sixpence")
rep <- ReplaceChar(x, " ", ".")
Print(rep)
# Result: Sing.a.song.of.sixpence

x <- c("sing a song of sixpence")
rep <- ReplaceChar(x, list("s", "x"), "p")
Print(rep)
# Result: ping a pong of pippence
```

See Also: for list items: `Replace()`, `SplitString()`

---

Name/Symbol: `ResetCanvas()`

Description: Resets a canvas, so that anything drawn onto it is erased and returned to its background color. Implemented by resetting the background color to itself:

```r
canvas.color <- canvas.
```

The function does not return the canvas, but has the side effect of resetting it.

Usage: `ResetCanvas(<list>)`
Example:

```lisp
# create a canvas, add pixel noise, then reset and repeat.
define Start(p)
{
  gWin <- MakeWindow()
  canvas <- MakeCanvas(100,100,MakeColor("black"))
  AddObject(canvas,gWin); Move(canvas,300,300)
  Draw()
  white <- MakeColor("white")
  ## add pixel noise
  j <- 1
  while(j < 5)
  {
    i <- 1
    while(i < 200)
    {
      SetPixel(canvas,Round(Random()*100),
               Round(Random()*100),white)
      i <- i +1
    }
  }
  Draw()
  WaitForAnyKeyPress()
  ResetCanvas(canvas)
  Draw()
  j <- j + 1
}
WaitForAnyKeyPress()
}
```

See Also: `+SetPixel()`, `+MakeCanvas()`, `+Draw()`

---

**Name/Symbol:** Rest()

**Description:** Returns the 'rest' of a list; a list minus its first element. If the list is empty or has a single member, it will return an empty list `[]`. This is a very common function in LISP.

**Usage:** Rest(<list>)

**Example:**
```lisp
x <- Sequence(1,5,1)
y <- Rest(x)
Print(rep)
# Result: [2,3,4,5]
```

**See Also:** Insert()
Name/Symbol: **RGBtoHSV()**

**Description:** Converts a color object to HSV values. May be useful for computing color-space distances and so on. No HSVtoRGB is currently implemented.

**Usage:** `RGBtoHSV(<color>)`

**Example:**
```
x <- RGBtoHSV(MakeColor("red"))
```

**See Also:** `MakeColor()`, `MakeColorRGB`

---

Name/Symbol: **return**

**Description:** Enables a function to return a value.

**Usage:**
```
define funcname()
{
    return 0
}
```

**Example:**

**See Also:**

---

Name/Symbol: **Rotate()**

**Description:** Returns a list created by rotating a list by `<n>` items. The new list will begin with the `<n+1>`th item of the old list (modulo its length), and contain all of its items in order, jumping back to the beginning and ending with the `<n>`th item. `Rotate(<list>,0)` has no effect. `Rotate` does not modify the original list.

**Usage:** `Rotate(<list-of-items>, <n>)`

**Example:**
```
Rotate([1,11,111],1) # == [11,111,1]
```

**See Also:** `Transpose()`

---

Name/Symbol: **RotatePoints**

215
**Description:** Takes a set of points (defined in a joined list \([x_1,x_2,x_3,\ldots], [y_1,y_2,y_3,\ldots]\) and rotates them \(<\text{angle}>\) degrees around the point \([0,0]\), returning a similar \([x],[y]\) list.

**Usage:** \(\text{ZoomPoints(<points>,<angle>)}\)

**Example:**

```
points <- \([1,2,3,4], [20,21,22,23]\)
newpoints <- \text{RotatePoints(points,10)}
```

**See Also:** \(\text{ZoomPoints()}, \text{ReflectPoints}\)

---

**Name/Symbol:** \(\text{Round()}\)

**Description:** Rounds \(<\text{num}>\) to nearest integer, or if optional \(<\text{precision}>\) argument is included, to nearest \(10^{-\text{precision}}\).

**Usage:**

```
\text{Round(<num>)}
\text{Round(<num>,<precision>)}
```

**Example:**

```
\text{Round(33.23)} \# == 33
\text{Round(56.65)} \# == 57
\text{Round(33.12234,2)} \# == 33.12
\text{Round(43134.23,-2)} \# == 43100
```

**See Also:** \(\text{Ceiling()}, \text{Floor()}, \text{AbsFloor()}, \text{ToInt()}\)
Chapter 8. Detailed Function and Keyword Reference

8.19 S

Name/Symbol: Sample()

Description: Samples a single item from a list, returning it. It is a bit more convenient at times than ShuffleN(list,1), which returns a list of length 1. Implemented as First(ShuffleN(list,1))

Usage: Sample(<list>)

Example: Sample([1,1,1,2,2]) # Returns a single number
Sample([1,2,3,4,5,6,7]) # Returns a single number

See Also: SeedRNG(), SampleNWithReplacement(), Subset()

Name/Symbol: SampleN()

Description: Samples <number> items from list, returning a randomly-ordered list. Items are sampled without replacement, so once an item is chosen it will not be chosen again. If <number> is larger than the length of the list, the entire list is returned shuffled. It differs from ChooseN in that ChooseN returns items in the order they appeared in the original list. It is implemented as Shuffle(ChooseN()).

Usage: SampleN(<list>, <n>)

Example: SampleN([1,1,1,2,2], 5) # Returns 5 numbers
SampleN([1,2,3,4,5,6,7], 3) # Returns 3 numbers

See Also: ChooseN(), SampleNWithReplacement(), Subset()

Name/Symbol: SampleNWithReplacement()

Description: SampleNWithReplacement samples <number> items from <list>, replacing after each draw so that items can be sampled again. <number> can be larger than the length of the list. It has no side effects on its arguments.

Usage: SampleNWithReplacement(<list>, <number>)
Example:  
```r
x <- Sequence(1:100,1)
SampleNWithReplacement(x, 10)
# Produces 10 numbers between 1 and 100, possibly
# repeating some.
```

See Also:  
SampleN(), ChooseN(), Subset()

---

**Name/Symbol:** SetProperty  
**Description:** Sets a property of a custom object. This works for custom or built-in objects, but new properties can only be set on custom object. This function works essentially identically to the obj.property assignment, but it allows you to create property names from input. It is used extensively for the PEBL parameter setting.  

**Usage:**  
```r
SetProperty(obj,property, value)
```

**Example:**  
```r
obj <- MakeCustomObject("myobject")
obj.taste <- "buttery"
obj.texture <- "creamy"
SetProperty(obj,"flavor","tasty")
```

```r
list <- GetPropertyList(obj)
loop(i,list)
{
  if(PropertyExists(obj,i))
  {
    Print(i + ": " + GetProperty(obj,i))
  }
}
```

See Also:  
GetProperty, PropertyExists, GetPropertyList

MakeCustomObject, PrintProperties

---

**Name/Symbol:** SDTBeta()  
**Description:** SDTBeta computes beta, as defined by signal detection theory.  

**Usage:**  
```r
SDTBeta(<hr>, <far>)
```

**Example:**  
```r
Print(SDTBeta(.1,.9)) #.67032
Print(SDTBeta(.1,.5)) #.88692
Print(SDTBeta(.5,.5)) #1
```
**Chapter 8. Detailed Function and Keyword Reference**

```r
Print(SDTBeta(.8,.9)) #0.918612
Print(SDTBeta(.9,.95)) #0.954803
```

Name/Symbol: **SaveAudioToWaveFile**

Description: Saves a buffer, recorded using the GetAudioInputBuffer, to a .wav file for later analysis or archive.

Usage: `SaveAudioToWaveFile(filename, buffer)`

This will save a .wav file of a buffer that was recorded (e.g., using GetVocalResponseTime).

See number-stroop.pbl in the stroop directory of the test battery and testaudioin.pbl in demo/ for examples.

Example:

```r
gResponseBuffer <- MakeAudioInputBuffer(5000)
resp0 <- GetVocalResponseTime(gResponseBuffer,.35, 200)
SaveAudioToWaveFile("output.wav",gResponseBuffer)
```

See Also: GetVocalResponseTime(), MakeAudioInputBuffer()

See Also: **SDTDPrime()**

```r
Name/Symbol: SDTDPrime()
```

Description: **SDTDPrime** computes d-prime, as defined by signal detection theory. This is a measure of sensitivity based jointly on hit rate and false alarm rate.

Usage: `SDTDPrime(<hr>, <far>)`

Example:

```r
Print(SDTDPri"me(.1,.9)) #2.56431
Print(SDTDPri"me(.1,.5)) #1.28155
Print(SDTDPri"me(.5,.5)) #0
Print(SDTDPri"me(.8,.9)) #.43993
Print(SDTDPri"me(.9,.95)) #.363302
```

See Also: SDTBeta()
Name/Symbol: SetCheckbox()

Description: This sets the .status property of a checkbox and draws it. Its state can also be updated using the the ClickCheckBox() function, which flips the current state.

Usage: SetCheckBox(obj, value)

Example:
```r
ok <- MakeCheckbox("OK?",400,250,gWin,150)
Draw()
SetCheckBox(ok,1)
Draw()
Wait(1000)
SetCheckbox(ok,0)
Draw()
Wait(1000)
```

Examples of its use can be found in demo ui.pbl

See Also: MakeCheckBox(), ClickCheckBox()

Name/Symbol: SeedRNG()

Description: Seeds the random number generator with <num> to reproduce a random sequence. This function can be used cleverly to create a multi-session experiment: Start by seeding the RNG with a single number for each subject; generate the stimulus sequence, then extract the appropriate stimuli for the current block. Remember to RandomizeTimer() afterward if necessary.

Usage: SeedRNG(<num>)

Example:
```r
##This makes sure you get the same random order
## across sessions for individual subjects.
SeedRNG(gSubNum)
stimTmp <- Sequence(1:100,1)
stim <- Shuffle(stimTmp)
RandomizeTimer()
```

See Also: RandomizeTimer()

Name/Symbol: SendData()

Description: Sends data on network connection. Example of usage in demo/nim.pbl. You can only send text data.
**Chapter 8. Detailed Function and Keyword Reference**

Usage: \(\text{SendData}(<\text{network}>,<\text{data}\_\text{as}\_\text{string}>)\)

Example: On 'server':

\[
\text{net} \leftarrow \text{WaitForNetworkConnection}(\text{"localhost"},1234) \\
\text{SendData}(\text{net},\text{"Watson, come here. I need you."}) \\
\text{CloseNetworkConnection}(\text{net})
\]

On Client:

\[
\text{net} \leftarrow \text{ConnectToHost}(\text{"localhost"},1234) \\
\text{value} \leftarrow \text{GetData}(\text{net},20) \\
\text{Print}(\text{value}) \\
\text{CloseNetworkConnection}(\text{net})
\]

##should print out "Watson, come here. I need you."

See Also: ConnectToIP, ConnectToHost, WaitForNetworkConnection, GetData, ConvertIPString, CloseNetworkConnection

---

**Name/Symbol:** SegmentsIntersect()

**Description:** Determines whether two line segments, defined by four xy point pairs, intersect. Two line segments that share a corner return 0, although they could be considered to intersect.

This function is defined in pbl-lib/Graphics.pbl

Usage: \(\text{SegmentsIntersect}(x1,y1,x2,y2, \\
a1,b1,a2,b2)\)

Example: \(\text{SegmentsIntersect}(1,0,2,0, \\
1,2,2,2) \#0\)

#returns 0, though they share (1,0)

\(\text{SegmentsIntersect}(1,0,2,0, \\
1,0,2,2)\)

\(\text{SegmentsIntersect}(1,1,3,1, \\
2,2,2,0) \#1\)

See Also: GetAngle3, ToRight

---

**Name/Symbol:** Sequence()

221
Description: Makes a sequence of numbers from `<start>` to `<end>` at `<step>`-sized increments. If `<step>` is positive, `<end>` must be larger than `<start>`, and if `<step>` is negative, `<end>` must be smaller than `<start>`. If `<start> + n*<step>` does not exactly equal `<end>`, the last item in the sequence will be the number closest number to `<end>` in the direction of `<start>` (and thus `<step>`).

Usage: `Sequence(<start>, <end>, <step>)`

Example:
- `Sequence(0,10,3)  # == [0,3,6,9]`
- `Sequence(0,10,1.5) # == [0,1.5,3,4.5, 6, 7.5, 9]`
- `Sequence(10,1,3)   # error`
- `Sequence(10,0,-1)  # == [10,9,8,7,6,5,4,3,2,1]`

See Also: `Repeat()`, `RepeatList()`

Name/Symbol: `SetCursorPosition()`

Description: Moves the editing cursor to a specified character position in a textbox.

Usage: `SetCursorPosition(<textbox>, <integer>)`

Example: `SetCursorPosition(tb, 23)`

See Also: `SetEditable()`, `GetCursorPosition()`, `SetText()`, `GetText()`

Name/Symbol: `SetEditable()`

Description: Sets the “editable” status of the textbox. All this really does is turns on or off the cursor; editing must be done with the (currently unsupported) device function `GetInput()`.

Usage: `SetEditable()`

Example: `SetEditable(tb, 0)`
`SetEditable(tb, 1)`

See Also: `GetEditable()`

Name/Symbol: `SetElement()`
**Chapter 8. Detailed Function and Keyword Reference**

**Description:** Efficiently alter a specific item from a list. `SetElement` has length-constant access time, and so it can be efficient to pre-create a list structure and then populate it one-by-one.

**Usage:**

```plaintext
SetElement(<list>, <index>, <value>)
```

**Example:**

```plaintext
##Set a random subset of elements to their index:
list <- Repeat(0,10)
index <- 1
while(index <= 10)
{
  if(Random()<.2)
  {
    SetElement(list,index,index)
  }
  index <- index + 1
}
```

**See Also:** `Nth()`, `Append()`, `PushOnEnd()`

---

**Name/Symbol:** `SetFont()`

**Description:** Resets the font of a textbox or label. Change will not appear until the next `Draw()` function is called. Can be used, for example, to change the color of a label to give richer feedback about correctness on a trial (see example below). Font can also be set by assigning to the `font` property of an object.

**Usage:**

```plaintext
SetFont(<text-widget>, <font>)
```

**Example:**

```plaintext
fontGreen <- MakeFont("vera.ttf",1,22,
  MakeColor("green"),
  MakeColor("black"), 1)
fontRed  <- MakeFont("vera.ttf",1,22,
  MakeColor("red"),
  MakeColor("black"), 1)
label <- MakeLabel(fontGreen, "Correct")
#Do trial here.
if(response == 1)
{
  setText(label, "CORRECT")
  SetFont(label, fontGreen)
}
```
} else {
    SetText(label, "INCORRECT")
    SetFont(label, "fontRed")
}
Draw()

See Also: SetText()

Name/Symbol: SetMouseCursorPosition()

Description: Sets the current x,y coordinates of the mouse pointer, 'warping' the mouse to that location immediately

Usage: SetMouseCursorPosition(<x>,<y>)

Example:
## Set mouse to center of screen:
SetMouseCursorPosition(gVideoWidth/2, gVideoHeight/2)

See Also: ShowCursor, WaitForMouseButton, SetMouseCursorPosition, GetMouseCursorPosition

Name/Symbol: SetPixel(), SetPoint()

Description: Sets the pixel at x,y to a particular color. It can also be called using SetPoint(). SetPoint is primarily useful for images and canvases—labels and textboxes get re-rendered upon draw so any use of SetPixel will get overwritten when it gets drawn. It won’t work on windows or shapes.

Usage: SetPixel(<x>,<y>,<color>)
SetPoint(<x>,<y>,<color>)

Example:
back <- MakeCanvas(50,50)
AddObject(back,gWin)
col <- MakeColor("green")
xy <- [[10,10],[10,11],[10,12],[10,13]]
loop(i,xy)
{
    SetPixel(First(i),Second(i),col)
}
Draw()

See Also: SetPoint, MakeGabor

224
Name/Symbol: SetPPortMode
Description: Sets a parallel port mode, either "<input>" or "<output>".
Usage: SetPPortMode("<input>")
Example:
See Also: COMPortGetByte, COMPortSendByte, OpenPPort OpenCOMPort, SetPPortMode, GetPPortState

Name/Symbol: SetPPortState
Description: Sets a parallel port state, using a list of 8 'bits' (1s or 0s).
Usage: SetPPortState([0,0,0,0,0,0,0,0])
Example:
See Also: COMPortGetByte, COMPortSendByte, OpenPPort OpenCOMPort, SetPPortMode, GetPPortState

Name/Symbol: SetScrollingText()
Description: This updates the text in a ScrollingTextBox. Because text must be parsed to be put into the box, you cannot just update the .text property, but instead should use this function.
Usage: SetScrollingText(stb, newtext)

Here, stb is a scrolling textbox created with MakeScrollingTextBox, and newtext is the new text you want to display.
Example: See uip.pbl in the demo directory for examples of the use of a scrolling text box. A brief example follows:

```
textscroll <- MakeScrollingTextBox("",200,50,gWin,12,300,150,0)
SetScrollingText(textscroll,FileReadText("Uppercase.txt"))
Draw()
resp <- WaitForClickOntarget([textscroll],[1])
CallFunction(textscroll.clickon,[textscroll,gClick])
```
**Chapter 8. Detailed Function and Keyword Reference**

See Also: MakeScrollingTextBox MakeScrollBox UpdateScrollBox DrawScrollBox ClickOnScrollBox

---

**Name/Symbol:** SetText()

**Description:** Resets the text of a textbox or label. Change will not appear until the next Draw() function is called. The object.text property can also be used to change text of an object, by doing: object.text <- "new text"

**Usage:** SetText(<text-widget>, <text>)

**Example:**

```r
# Fixation Cross:
label <- MakeLabel(font, "+")
Draw()

SetText(label, "X")
Wait(100)
Draw()
```

See Also: GetText(), SetFont()

---

**Name/Symbol:** Show()

**Description:** Sets a widget to visible, once it has been added to a parent widget. This just changes the visibility property, it does not make the widget appear. The widget will not be displayed until the Draw() function is called. The .visible property of objects can also be used to hide or show the object.

**Usage:** Show(<object>)

**Example:**

```r
window <- MakeWindow()
image1 <- MakeImage("pebl.bmp")
image2 <- MakeImage("pebl.bmp")
AddObject(image1, window)
AddObject(image2, window)
Hide(image2)
Draw()
Wait(300)
Show(image2)
Draw()
```

See Also: Hide()
**Chapter 8. Detailed Function and Keyword Reference**

**Name/Symbol:** ShowCursor()

**Description:** Hides or shows the mouse cursor. Currently, the mouse is not used, but on some systems in some configurations, the mouse cursor shows up. Calling ShowCursor(0) will turn off the cursor, and ShowCursor(1) will turn it back on. Be sure to turn it on at the end of the experiment, or you may actually lose the cursor for good.

**Usage:**

```
ShowCursor(<value>)
```

**Example:**

```
window <- MakeWindow()
ShowCursor(0)
## Do experiment here
##
## Turn mouse back on.
ShowCursor(1)
```

**See Also:**

**Name/Symbol:** Shuffle()

**Description:** Randomly shuffles a list.

**Usage:**

```
Shuffle(list)
```

**Example:**

```
Print(Shuffle([1,2,3,4,5]))
# Results might be anything, like [5,3,2,1,4]
```

**See Also:** Sort(), SortBy(), ShuffleRepeat(), ShuffleWithoutAdjacents()

**Name/Symbol:** ShuffleRepeat()

**Description:** Randomly shuffles <list>, repeating <n> times. Shuffles each iteration of the list separately, so you are guaranteed to go through all elements of the list before you get another. Returns a nested list.

**Usage:**

```
ShuffleRepeat(<list>, <n>)
```

**Example:**

```
Print(ShuffleRepeat([1,2,3,4,5]),3)
## Results might be anything, like:
##    [[5,3,2,1,4], [3,2,5,1,4], [1,4,5,3,2]]
```

Typically, you will want to flatten before using:
list <- Flatten(ShuffleRepeat([1,2,3], 5))

See Also: Sort(), SortBy(), ShuffleRepeat(), ShuffleWithoutAdjacents()

Name/Symbol: ShuffleWithoutAdjacents()

Description: Randomly shuffles <nested-list>, attempting to create a list where the nested elements do not appear adjacent in the new list. Returns a list that is flattened one level. It will always return a shuffled list, but it is not guaranteed to return one that has the non-adjacent structure specified, because this is sometimes impossible or very difficult to do randomly. Given small enough non-adjacent constraints with enough fillers, it should be able to find something satisfactory.

Usage: ShuffleWithoutAdjacents(<nested-list>)

Example:

```
print(ShuffleWithoutAdjacents([[1,2,3], [4,5,6], [7,8,9]]))
```

## Example Output:
## [8, 5, 2, 7, 4, 1, 6, 9, 3]
## [7, 4, 8, 1, 9, 2, 5, 3, 6]

## Non-nested items are shuffled without constraint
print(ShuffleWithoutAdjacents([[1,2,3], [11,12,13,14,15,16]]))

## output: [13, 11, 2, 14, 3, 15, 1, 16, 12]
## [13, 12, 2, 16, 15, 11, 1, 14, 3]
## [11, 1, 15, 2, 12, 16, 14, 13, 3]

## Sometimes the constraints cannot be satisfied.
## 9 will always appear in position 2
print(ShuffleWithoutAdjacents([[1,2,3], 9]))

## output: [3, 9, 1, 2]
## [2, 9, 3, 1]
## [3, 9, 2, 1]

See Also: Shuffle(), Sort(), SortBy(), ShuffleRepeat(), ShuffleWithoutAdjacents()

Name/Symbol: Sign()

Description: Returns +1 or -1, depending on sign of argument.
**Chapter 8. Detailed Function and Keyword Reference**

Usage: \( \text{Sign}(\text{num}) \)

Example:
- \( \text{Sign}(-332.1) \) # == -1
- \( \text{Sign}(65) \) # == 1

See Also: \( \text{Abs()} \)

---

**Name/Symbol:** \( \text{SignalFatalError()} \)

Description: Stops PEBL and prints \(<\text{message}>\) to stderr. Useful for type-checking in user-defined functions.

Usage: \( \text{SignalFatalError}(\text{<message>}) \)

Example:
```pseudocode
If(\text{not IsList(x)})
{
    \text{SignalFatalError}("Tried to frobnicate a List.")
}
# Prints out error message and
# line/filename of function
```

See Also: \( \text{Print()} \)

---

**Name/Symbol:** \( \text{Sin()} \)

Description: Sine of \(<\text{deg}>\) degrees.

Usage: \( \text{Sin}(\text{<deg>}) \)

Example:
- \( \text{Sin}(180) \)
- \( \text{Sin}(0) \)

See Also: \( \text{Cos()}, \text{Tan()}, \text{ATan()}, \text{ACos()}, \text{ATan}() \)

---

**Name/Symbol:** \( \text{Sort()} \)

Description: Sorts a list by its values from smallest to largest.

Usage: \( \text{Sort}(\text{<list>}) \)

Example:
- \( \text{Sort}([3,4,2,1,5]) \) # == [1,2,3,4,5]

See Also: \( \text{SortBy()}, \text{Shuffle()} \)
Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: **SortBy()**

Description: Sorts a list by the values in another list, in ascending order.

Usage: \[\text{SortBy(<value-list>, <key-list>)}\]

Example: \[
\text{SortBy(["Bobby","Greg","Peter"], [3,1,2])}
\]

# == ["Greg","Peter","Bobby"]

See Also: Shuffle(), Sort()

Name/Symbol: **SplitString()**

Description: Splits a string into tokens. \(<\text{split}>\) must be a string. If \(<\text{split}>\) is not found in \(<\text{string}>\), a list containing the entire string is returned; if \(\text{split}\) is equal to "", the each letter in the string is placed into a different item in the list. Only the first character of \(<\text{split}>\) is used. If you need a multi-character \(\text{split}\), you can use \(<\text{SplitStringSlow}>\), which can handle multi-character \(\text{split}\)s but is relatively slower. This should not matter for short strings, but if you are using \(\text{SplitString}\) on long files, it could make a difference.

Usage: \[\text{SplitString(<string>, <split>)}\]

Example: \[
\text{SplitString("Everybody Loves a Clown", " ")}
\]

# Produces ["Everybody", "Loves", "a", "Clown"]

See Also: FindInString(), ReplaceChar, SplitStringSlow

Name/Symbol: **SplitStringSlow()**

Description: Splits a string into tokens. \(<\text{split}>\) must be a string. If \(<\text{split}>\) is not found in \(<\text{string}>\), a list containing the entire string is returned; if \(\text{split}\) is equal to "", the each letter in the string is placed into a different item in the list. The entire text of \(<\text{split}>\) is used to tokenize, but as a consequence this function is relatively slow, and should be avoided if your string is longer than a few hundred characters.

Usage: \[\text{SplitStringSlow(<string>, <split>)}\]
Chapter 8. Detailed Function and Keyword Reference

Example:    
\[
\text{SplitStringSlow("Everybody Loves a Clown", " ")}
\]  
\# Produces ["Everybody", "Loves", "a", "Clown"]

\[
\text{SplitStringSlow("she sells seashells", "ll")}
\]  
\#produces ["she se", "s seashe", "s"]

See Also:  
\[
\text{Splitstring FindInString(), ReplaceChar}
\]

Name/Symbol:  
Square()

Description:  
Creates a square for graphing at \(x, y\) with size \(<\text{size}>\). Squares are only currently definable oriented in horizontal/vertical directions. A square must be added to a parent widget before it can be drawn; it may be added to widgets other than a base window. The properties of squares may be changed by accessing their properties directly, including the FILLED property which makes the object an outline versus a filled shape.

Usage:  
\[
\text{Ellipse(<x>, <y>, <size>, <color>)}
\]

Example:  
\[
\text{s <- Square(30,30,20, MakeColor(green))}
\text{AddObject(s, win)}
\text{Draw()}
\]

See Also:  
\[
\text{Circle(), Ellipse(), Rectangle(), Line()}
\]

Name/Symbol:  
Sqrt()

Description:  
Square root of \(<\text{num}>\).

Usage:  
\[
\text{Sqrt(<num>)}
\]

Example:  
\[
\text{Sqrt(100) \# == 10}
\]

See Also:

Name/Symbol:  
StartEventLoop()

Description:  
Starts the event loop with currently-registered events. This function is currently experimental, and its usage may change in future versions of PEBL.

Usage:  
\[
\text{StartEventLoop()}\]

231
Example:

See Also: RegisterEvent(), ClearEventLoop()

Name/Symbol: StartPlayback()

Description: Initiates playback of a movie so that it will play in the background when a Wait() or WaitFor() function is called. This allows one to collect a response while playing a movie. The movie will not actually play until the event loop is started, typically with something like Wait().

Usage: StartPlayback(movie)

Example:

movie <- LoadMovie("movie.avi", gWin, 640, 480)
PrintProperties(movie)
Move(movie, 20, 20)
Draw()
StartPlayback(movie)
Wait(500) # Play 500 ms of the movie.
PausePlayback(movie)

See Also: LoadAudioFile(), LoadMovie(), PlayMovie(), PausePlayback()

Name/Symbol: StDev()

Description: Returns the standard deviation of <list>.

Usage: StDev(<list>)

Example:

sd <- StDev([3, 5, 99, 12, 1.3, 15])

See Also: Min(), Max(), Mean(), Median(), Quantile(), Sum()

Name/Symbol: Stop()

Description: Stops a sound playing in the background from playing. Calling Stop() on a sound object that is not playing should have no effect, but if an object is aliased, Stop() will stop the file. Note that sounds play in a separate thread, so interrupting the thread has a granularity up to the duration of the thread-switching quantum on your computer; this may be tens of milliseconds.

Usage: Stop(<sound-object>)
Chapter 8. Detailed Function and Keyword Reference

Example:

```r
buzz <- LoadSound("buzz.wav")
PlayBackground(buzz)
Wait(50)
Stop(buzz)
```

See Also: `PlayForeground()`, `PlayBackground()`

---

Name/Symbol: `StringLength()`

Description: Determines the length of a string, in characters.

Usage: `StringLength(<string>)`

Example:

```r
StringLength("absolute")  # == 8
StringLength("  spaces ")  # == 12
StringLength(""")  # == 0
```

See Also: `Length()`, `SubString()`

---

Name/Symbol: `StripQuotes()`

Description: Strips quotation marks from the outside of a string. Useful if you are reading in data that is quoted.

Usage: `StripQuotes(<text>)`

Example:

```r
text <- gQuote + "abcd" + gQuote
Print(StripQuotes(text))  ## abcd
Print(StripQuotes("aaa"))  ## aaa
```

See Also: `StripSpace()`

---

Name/Symbol: `StripSpace()`

Description: Strips spaces from the start and end of a string. Useful for cleaning up input and such.

Usage: `StripSpaces(<text>)`

Example:

```r
text <- " abc d 
Print(StripSpace(text))  ## 'abcd'
Print(StripSpace("aaa"))  ## 'aaa'
```

See Also: `StripQuotes()`

---

233
Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: SubList()
Description: Extracts a list from another list, by specifying beginning and end points of new sublist.
Usage: SubList(<list>, <begin>, <end>)
Example: SubList([1,2,3,4,5,6],3,5) # == [3,4,5]
See Also: SubSet(), ExtractListItems()

Name/Symbol: Subset()
Description: Extracts a subset of items from another list, returning a new list that includes items from the original list only once and in their original orders. Item indices in the second argument that do not exist in the first argument are ignored. It has no side effects on its arguments.
Usage: Subset(<list>, <list-of-indices>)
Example: Subset([1,2,3,4,5,6],[5,3,1,1]) # == [1,3,5]
Subset([1,2,3,4,5], [23,4,2]) # == [2,4]
See Also: SubList(), ExtractItems(), SampleN()

Name/Symbol: SubString()
Description: Extracts a substring from a longer string.
Usage: SubString(<string>,<position>,<length>)
Example: SubString("abcdefghijklmnop",3,5) # == "cdefg"
See Also:

Name/Symbol: Sum()
Description: Returns the sum of <list>.
Usage: Sum(<list>)
Example: \( \text{sum} \leftarrow \text{StDev([3,5,99,12,1.3,15])} \) \# == 135.3

See Also: \text{Min()}, \text{Max()}, \text{Mean()}, \text{Median()}, \text{Quantile()}, \text{StDev()}

Name/Symbol: \text{SummaryStats()}

Description: Computes summary statistics for a data list, aggregated by labels in a condition list. For each condition (distinct label in the \(<\text{cond}>\) list), it will return a list with the following entries: \(<\text{cond}>\) \(<\text{N}>\) \(<\text{median}>\) \(<\text{mean}>\) \(<\text{sd}>\)

Usage: \text{SummaryStats(<data>,<cond>)}

Example: \( \text{dat} \leftarrow [1.1, 1.2, 1.3, 2.1, 2.2, 2.3] \)
\( \text{cond} \leftarrow [1, 1, 1, 2, 2, 2] \)
\( \text{Print(SummaryStats(dat, cond))} \)

Result:
\[
[[1, 3, 1.1, 1.2, 0.0816497], [2, 3, 2.1, 2.2, 0.0816497]]
\]

See Also: \text{StDev()}, \text{Min()}, \text{Max()}, \text{Mean()}, \text{Median()}, \text{Quantile()}, \text{Sum()}

Name/Symbol: \text{SystemCall()}

Description: Calls/runs another operating system command. Can also be used to launch another PEBL program. Useful to check \text{GetSystemType()} before running.

Note that the output of a command-line argument is generally not passed back into PEBL; just the function's return code, which is usually 0 on success or some other number on failure (depending upon the type of failure). Some uses might include:

Usage: \text{SystemCall("text-of-command")}
\text{SystemCall("text-of-command","command-line-options")}

Example: \( \text{if(GetSystemType() == "WINDOWS")} \)
\{ \( x \leftarrow \text{SystemCall("dir input.txt")} \) \} \text{ else } \{ \( x \leftarrow \text{SystemCall("ls input.txt")} \) \}
\( \text{if(x <> 0)} \)
Chapter 8. Detailed Function and Keyword Reference

{  
    SignalFatalError("Expected file "+
              "input.txt] does not exist")
}

See Also: GetSystemType()
Chapter 8. Detailed Function and Keyword Reference

8.20 T

Name/Symbol: Tab()
Description: Produces a tab character which can be added to a string. If displayed in a text box, it will use a 4-item tab stop.
Usage: Tab(3)
Example:
Print("Number: ", Tab(1) + number )
Print("Value: ", Tab(1) + value )
Print("Size: ", Tab(1) + size )
See Also: Format(), CR()

Name/Symbol: Tan()
Description: Tangent of <deg> degrees.
Usage: Tan(<deg>)
Example: Tan(180)
See Also: Cos(), Sin(), ATan(), ACos(), ATan()

Name/Symbol: ThickLine()
Description: Makes a thick line between two coordinates. This just creates a polygon object to serve as the line.
Usage: ThickLine(<x1>,<y1>,<x2>,<y2>, <size-in-pixels>,<color>)
Example:
    a <- ThickLine(10,10,300,400,20,
               MakeColor("red"))
    AddObject(a,gWin)
    Draw()
See Also: Line(), Polygon()

Name/Symbol: TimeStamp()
Chapter 8. Detailed Function and Keyword Reference

Description: Returns a string containing the date-and-time, formatted according to local conventions. Should be used for documenting the time-of-day and date an experiment was run, but not for keeping track of timing accuracy. For that, use `GetTime()`.

Usage: `TimeStamp()`

Example:
```i
a <- TimeStamp()
Print(a)
```

See Also: `GetTime()`

Name/Symbol: `ToInt()`

Description: Rounds a number to an integer, changing internal representation.

Usage:
```i
ToInt(<number>)
ToInt(<floating-point>)
ToInt(<string-as-number>)
```

Example:
```i
ToInt(33.332) # == 33
ToInt("3213") # == 3213
```

See Also: `Round()`, `Ceiling()`, `AbsCeiling()`, `Floor()`, `AbsFloor()`

Name/Symbol: `ToFloat()`

Description: Converts number to internal floating-point representation.

Usage: `ToFloat(<number>)`

Example:

See Also:

Name/Symbol: `ToNumber()`

Description: Converts a variant to a number. Most useful for character strings that are interpretable as a number, but may also work for other subtypes.

Usage:
```i
ToNumber(<string>)
ToNumber(<number>)
```
Example:  
```
a <- ToNumber("3232")
Print(a + 1) # produces the output 3233.
```

See Also:  
`ToString()`, `ToFloa()` , `Round()`

### Name/Symbol: ToRight()

**Description:** Determines whether a point p3 is 'to the right' of a line segment defined by p1 to p2. Works essentially by computing the determinant.

**Usage:**  
```
ToRight(<p1>,<p2>,<p3>)
```

**Example:**  
```
a <- [100,0]
b <- [100,100]
c <- [150,50]
ToRight(a,b,c) # returns 1; true
ToRight(b,a,c) # returns 0; false
```

See Also:  
`GetAngle()`, `GetAngle3`, `SegmentsIntersect`

### Name/Symbol: ToString()

**Description:** Converts value to a string representation. Most useful for numerical values. This conversion is done automatically when strings are combined with numbers.

**Usage:**  
```
ToString(<number>)
ToString(<string>)
```

**Example:**  
```
a <- ToString(333.232)
Print(a + "111")
# produces the output '333.232111'.
```

See Also:  
`ToString()`, `+`

### Name/Symbol: TranslateKeyCode()

**Description:** Translates a code corresponding to a keyboard key into a keyboard value. This code is returned by some event/device polling functions.

**Usage:**
Example:

See Also:

Name/Symbol: Transpose()
Description: Transposes or "rotates" a list of lists. Each sublist must be of the same length.
Usage: Transpose(<list-of-lists>)
Example: Transpose([[1,11,111],[2,22,222],
                    [3,33,333],[4,44,444]])
         # == [[1,2,3,4],[11,22,33,44],
         #      [111,222,333,444]]
See Also: Rotate()
Name/Symbol: UpdatePulldown()

Description: This changes the list being used in a Pulldown object. It tries to maintain the same selected option (matching the text of the previous selection), but if not found will select index 1. It calls DrawPullDown when complete, but a Draw() command must be issued before the pulldown changes will appear.

Usage: UpdatePulldown(object, newlist)

Example:

options <- MakePulldownButton(["A", "B", "C"], 400, 250, gWin, 14, 100, 3)
Draw()
WaitForAnyKeyPress()

##This should add a fourth option but C should still be selected.
UpdatePulldown(options, ["A", "B", "C", "D"])
Draw()
WaitForAnyKeyPress()

See Also: MakePullDown(), Pulldown(), DrawPullDown

Name/Symbol: UpdateScrollbox()

Description: This updates the layout of a ScrollBox. It should be used if you manually change the .list or .listoffset properties. It won’t actually redraw the scrollbar (which is done by DrawScrollbox).

Usage: UpdateScrollBox(sb)

Here, sb is the scrollbox object.

Example: See ui.pbl in the demo directory for examples of the use of a scrolling text box. A brief example follows:

241
Chapter 8. Detailed Function and Keyword Reference

sb <- MakeScrollBox(Sequence(1,50,1),"The numbers",40,40,gWin,12,150,5)
Draw()

resp <- WaitForClickOntarget([sb],[1])
CallFunction(sb.clickon,[sb,gClick])
#Alternately: ClickOnScrollbox(sb,gClick)

##change the selected items
sb.list <- Sequence(sb.selected,sb.selected+50,1)
UpdateScrollbox(sb)
DrawScrollbox(sb)
Draw()

See Also: MakeScrollingTextBox MakeScrollBox DrawScrollView

Name/Symbol: Uppercase()

Description: Changes a string to uppercase. Useful for testing user input
against a stored value, to ensure case differences are not de-
tected.

Usage: Uppercase(<string>)

Example: Uppercase("POtaTo") # == "POTATO"

See Also: Lowercase()
Name/Symbol: Wait()
Description: Waits the specified number of milliseconds, then returns.
Usage: Wait(<time>)
Example: Wait(100)
Wait(15)
See Also:

Name/Symbol: WaitForAllKeysUp()
Description: Wait until all keyboard keys are in the up position. This includes numlock, capslock, etc.
Usage:
Example:
See Also:

Name/Symbol: WaitForAnyKeyDown()
Description: Waits for any key to be detected in the down position. This includes numlock, capslock, etc, which can be locked in the down position even if they are not being held down. Will return immediately if a key is being held down before the function is called.
Usage:
Example:
See Also: WaitForAnyKeyPress()

Name/Symbol: WaitForAnyKeyDownWithTimeout()
Description: Waits until any key is detected in the down position, but will return after a specified number of milliseconds.
Usage: WaitForAnyKeyDownWithTimeout(<time>)
**Chapter 8. Detailed Function and Keyword Reference**

Example:

See Also:

---

**Name/Symbol:** WaitForClickOnTarget()

**Description:** Allows you to specify a list of graphical objects in `<objectlist>` and awaits a click on any one of them, returning the corresponding key in `<keylist>`. Also, sets the global variable `gClick` which saves the location of the click, if you need it for something else.

**Usage:**

```r
x <- WaitForClickOnTarget(<objectlist>,<keylist>)
```

**Example:**

```r
click <- WaitForClickOnTarget(objs,resp)
```

**See Also:**

---

**Name/Symbol:** WaitForClickOnTargetWithTimeout()

**Description:** Allows you to specify a list of graphical objects in `<objectlist>` and awaits a click on any one of them, returning the corresponding key in `<keylist>`. Also, sets the global variable `gClick` which saves the location of the click, if you need it for something else. The function will return after the specified time limit.

If no response is made by timeout, the text `<timeout>` will be returned (instead of the corresponding keylist element), and `gClick` will be set to [-1, -1].

This function can also be useful to dynamically update some visual object while waiting for a response. Give timeout some small number (below 50 ms, as low as 1-5), and loop over this repeatedly until a 'proper' response is given, redrawing a timer or other dynamic visual element each time.
Chapter 8. Detailed Function and Keyword Reference

Usage:

\[
x <- \text{WaitForClickOnTarget}(<\text{objectlist}>, <\text{keylist}>, <\text{timeout-in-ms}>)
\]

Example:

\[
\begin{align*}
\text{resp} &<- \text{Sequence}(1,5,1) \\
\text{objs} &<- [] \\
\text{loop}(i,\text{resp}) \\
&\{ \\
&\quad \text{tmp} <- \text{EasyLabel}(i + ", ", 100+50*i,100,gWin,25) \\
&\quad \text{objs} <- \text{Append}(\text{objs}, \text{tmp}) \\
&\} \\
\text{Draw}() \\
\text{click} &<- \text{WaitForClickOnTargetWithTimeout}(\text{objs}, \text{resp}, 3000) \\
\text{Print}("You clicked on ", \text{click}) \\
\text{Print}("Click location: [" + \text{First}(\text{gClick}) + ", ", " + \text{Second}(\text{gClick}) + "]")
\end{align*}
\]

See Also:

\text{WaitForDownClick()}, \text{WaitForMouseButton()}

---

Name/Symbol: \text{WaitForDownClick()}

Description: Will wait until the mouse button is clicked down. Returns the same 4-tuple as \text{WaitForMouseButton}:

\[
[x_{\text{pos}}, \\
y_{\text{pos}}, \\
\text{button id \{1-3\}}, \quad \\
"<\text{pressed}>" \text{ or } "<\text{released}>"
\]

but the last element will always be \text{<pressed>}. Useful as a 'click mouse to continue' probe.

Usage:

\text{WaitForDownClick()}

Example:

\[
\begin{align*}
x &<- \text{WaitForDownClick()} \\
\text{Print}("Click location: [" + \text{First}(x) + ", ", " + \text{Second}(x) + "]")
\end{align*}
\]

See Also:

\text{WaitForClickOnTarget()}, \text{WaitForMouseButton()}

---

Name/Symbol: \text{WaitForKeyListDown()}

Description: Returns when any one of the keys specified in the argument is down. If a key is down when called, it will return immediately.

Usage:

\text{WaitForKeyListDown(<list-of-keys>)}
Example: \texttt{WaitForKeyListDown(["a","z"])}

See Also:

Name/Symbol: \texttt{WaitForKeyListDown()}

Description: Returns when any one of the keys specified in the argument is pressed, or when the timeout has elapsed; whichever comes first. Will only return on a new keyboard/timeout events, and so a previously pressed key will not trip this function, unlike \texttt{WaitForKeyListDown()}. The \texttt{<style>} parameter is currently unused, but may be deployed in the future for differences in how or when things should be returned. Returns the value of the pressed key. If the function terminates by exceeding the \texttt{<timeout>}, it will return the string \texttt{["<timeout>"]}.

Usage: \texttt{WaitForKeyListDown([list-of-keys], timeout, <style>)}

\texttt{<list-of-keys>} can include text versions of many keys. See Chapter 4, section “Keyboard Entry” for complete list of key-names.

Example: \begin{verbatim} x <- WaitForListKeyPressWithTimeout(["a","z"], 2000,1)  if(IsList(x))  {  Print("Did Not Respond.")  } \end{verbatim}

See Also: \texttt{WaitForKeyListDown}, \texttt{WaitForKeyListKeyPressWithTimeout}

Name/Symbol: \texttt{WaitForKeyListKeyPress()} 

Description: Returns when any one of the keys specified in the argument is pressed. Will only return on a new keyboard event, and so a previously pressed key will not trip this function, unlike \texttt{WaitForKeyListDown()}. Returns a string indicating the value of the keypress.

Usage: \texttt{WaitForKeyListKeyPress([list-of-keys])}

Example: \texttt{WaitForKeyListKeyPress(["a","z"])}

See Also: \texttt{WaitForKeyListDown}, \texttt{WaitForKeyListKeyPressWithTimeout}
Chapter 8. Detailed Function and Keyword Reference

Name/Symbol: WaitForKeyPress()

Description: Waits for a keypress event that matches the specified key. Usage of this function is preferred over WaitForKeyDown(), which tests the state of the key. Returns the value of the key pressed.

Usage: WaitForKeyPress(<key>)

Example:

See Also: WaitForAnyKeyPress(), WaitForKeyRelease(), WaitForListKeyPress()

Name/Symbol: WaitForKeyUp()

Description:

Usage:

Example:

See Also:

Name/Symbol: WaitForMouseButton()

Description: Waits for a mouse click event to occur. This takes no arguments, and returns a 4-tuple list, indicating:

[xpos, ypos, button id [1-3], "<pressed>" or "<released>]"

Usage: WaitForMouseButton()

Example: ## Here is how to wait for a mouse down-click

continue <- 1
while(continue)
{
  x <- WaitForMouseButton()
  if(Nth(x,4)="<pressed>")
  {
    continue <- 0
  }
}
Print("Clicked")
See Also: ShowCursor, WaitForMouseButtonWithTimeout, SetMouseCursorPosition, GetMouseCursorPosition

Name/Symbol: WaitForMouseButtonWithTimeout()

Description: Waits for a mouse click event to occur, or a timeout to be reached. This takes a single argument: timeout delay in ms. When clicked, it returns a 4-tuple list, indicating:

[xpos, ypos, button id [1-3], "<pressed>" or "<released>"]

when not clicked and timeout is reached, it returns a list: [timeout]

Usage: WaitForMouseButtonWithTimeout()

Example: ## Here is how to wait for a mouse down-click

```plaintext
continue <- 1
while(continue)
{
  x <- WaitForMouseButtonWithTimeout()
  if(First(x)=="<timeout>"
  {
    Print("time is "+GetTime())
    continue <- 1
  } else {
    continue <- 0
  }
}
Print("Clicked")
```

See Also: ShowCursor, SetMouseCursorPosition, GetMouseCursorPosition

Name/Symbol: WaitForNetworkConnection()

Description: Listens on a port, waiting until another computer or process connects. Return a network object that can be used for communication.

Usage: WaitForNetworkConnection(<port>)
Example: See nim.pbl for example of two-way network connection.

```
net <- WaitForNetworkConnection(1234)
dat <- GetData(net,20)
Print(dat)
CloseNetworkConnection(net)
```

See Also: ConnectToHost, ConnectToIP, GetData, WaitForNetworkConnection, SendData, ConvertIPString, CloseNetworkConnection

---

Name/Symbol: `while`

Description: ‘while’ is a keyword, and so is part of the syntax, not a function per se. It executes the code inside the `{}` brackets until the test inside the () executes as false. This can easily lead to an infinite loop if conditions are not met. Also, there is currently no break statement to allow execution to halt early. Unlike some other languages, PEBL requires that the `{}` be present.

Usage:

```
while(<test expression>)
{
    code line 1
    code line 2
}
```

Example: `i <- 1`  
```
while(i <= 10)
{
    Print(i)
    i <- i + 1
} # prints out the numbers 1 through 10
```

See Also: `loop()`, `{ }`

---

Name/Symbol: `WritePNG()`

Description: `WritePNG()` creates a graphic file of the screen or a widget on the screen. It can also be given an arbitrary widget. For the most part, widgets added to other widgets will be captured fine, but sometimes polygons and shapes added to other widgets may not appear in the output png.

249
Usage: 
\[
x \leftarrow \text{WritePNG("screen1.png",gWin)}
\]

## Use like this to create an animated screencast

```r
define DrawMe()
{
    pname <- "fileout"+ZeroPad(gid,5)+".png"
    Draw()
    WritePNG(pname,gWin)
}
```

```r
define Start(p)
{
    gid <- 1
gWin <- MakeWindow()
img <- MakeImage("pebl.png")
AddObject(img,gWin)
while(gid < 100)
{
    Move(img,RandomDiscrete(800),
         RandomDiscrete(600))
    DrawMe()
    gid <- gid + 1
}
}
```

See Also: FileWriteTable
Name/Symbol: ZeroPad

Description: Takes a number and pads it with zeroes left of the decimal point so that its length is equal to <size>. Argument must be a positive integer and less than ten digits. Returns a string.

Usage: ZeroPad(<number>, <length>)

Example:
Print(ZeroPad(33,5)) # "00033"
Print(ZeroPad(123456,6)) #"123456"
Print(ZeroPad(1,8)) #"00000001"

See Also: Format()

Name/Symbol: ZoomPoints

Description: Takes a set of points (defined in a joined list [[x1,x2,x3,...],[y1,y2,y3,...]] and adjusts them in the x and y direction independently, returning a similar [[x],[y]] list.

Note: The original points should be centered at zero, because the get adjusted relative to zero, not relative to their center.

Usage: ZoomPoints(points,<xzoom>,<yzoom>)

Example:
points <- [[1,2,3,4],[20,21,22,23]]
newpoints <- ZoomPoints(points,2,5)
##Produces [[2,4,6,8],[10,11.5,11,11.5]]

See Also: RotatePoints(), ReflectPoints
Chapter 9
Color Name Reference

In PEBL, around 750 colors can be accessed by name, using the MakeColor() function. Each name corresponds to a specific RGB value. The following table provides examples of the particular color names, RGB values, and the obtained shade produced by PEBL.

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALICE BLUE</td>
<td>240</td>
<td>248</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>ALICEBLUE</td>
<td>240</td>
<td>248</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>ANTIQUE WHITE</td>
<td>250</td>
<td>235</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>ANTIQUEWHITE</td>
<td>250</td>
<td>235</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>ANTIQUEWHITE1</td>
<td>255</td>
<td>239</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>ANTIQUEWHITE2</td>
<td>238</td>
<td>223</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>ANTIQUEWHITE3</td>
<td>205</td>
<td>192</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>ANTIQUEWHITE4</td>
<td>139</td>
<td>131</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>AQUAMARINE</td>
<td>127</td>
<td>255</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>AQUAMARINE1</td>
<td>127</td>
<td>255</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>AQUAMARINE2</td>
<td>118</td>
<td>238</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>AQUAMARINE3</td>
<td>102</td>
<td>205</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>AQUAMARINE4</td>
<td>69</td>
<td>139</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>AZURE</td>
<td>240</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>AZURE1</td>
<td>240</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>AZURE2</td>
<td>224</td>
<td>238</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>AZURE3</td>
<td>193</td>
<td>205</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>AZURE4</td>
<td>131</td>
<td>139</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>BEIGE</td>
<td>245</td>
<td>245</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>BISQUE</td>
<td>255</td>
<td>228</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>BISQUE1</td>
<td>255</td>
<td>228</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>BISQUE2</td>
<td>238</td>
<td>213</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
<td>Example</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>BISQUE3</td>
<td>205</td>
<td>183</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>BISQUE4</td>
<td>139</td>
<td>125</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>BLACK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BLANCHED ALMOND</td>
<td>255</td>
<td>235</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>BLANCHEDALMOND</td>
<td>255</td>
<td>235</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>BLUE</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>BLUE VIOLET</td>
<td>138</td>
<td>43</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>BLUE1</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>BLUE2</td>
<td>0</td>
<td>0</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>BLUE3</td>
<td>0</td>
<td>0</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>BLUE4</td>
<td>0</td>
<td>0</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>BLUEVIOLET</td>
<td>138</td>
<td>43</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>BROWN</td>
<td>165</td>
<td>42</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>BROWN1</td>
<td>255</td>
<td>64</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>BROWN2</td>
<td>238</td>
<td>59</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>BROWN3</td>
<td>205</td>
<td>51</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>BROWN4</td>
<td>139</td>
<td>35</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>BURLYWOOD</td>
<td>222</td>
<td>184</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>BURLYWOOD1</td>
<td>255</td>
<td>211</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>BURLYWOOD2</td>
<td>238</td>
<td>197</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>BURLYWOOD3</td>
<td>205</td>
<td>170</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>BURLYWOOD4</td>
<td>139</td>
<td>115</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>CADET BLUE</td>
<td>95</td>
<td>158</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>CADETBLUE</td>
<td>95</td>
<td>158</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>CADETBLUE1</td>
<td>152</td>
<td>245</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>CADETBLUE2</td>
<td>142</td>
<td>229</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>CADETBLUE3</td>
<td>122</td>
<td>197</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>CADETBLUE4</td>
<td>83</td>
<td>134</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>CHARTREUSE</td>
<td>127</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CHARTREUSE1</td>
<td>127</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CHARTREUSE2</td>
<td>118</td>
<td>238</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CHARTREUSE3</td>
<td>102</td>
<td>205</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CHARTREUSE4</td>
<td>69</td>
<td>139</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CHOCOLATE</td>
<td>210</td>
<td>105</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>CHOCOLATE1</td>
<td>255</td>
<td>127</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>CHOCOLATE2</td>
<td>238</td>
<td>118</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>CHOCOLATE3</td>
<td>205</td>
<td>102</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>CHOCOLATE4</td>
<td>139</td>
<td>69</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>CORAL</td>
<td>255</td>
<td>127</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>CORAL1</td>
<td>255</td>
<td>114</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>CORAL2</td>
<td>238</td>
<td>106</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>CORAL3</td>
<td>205</td>
<td>91</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>CORAL4</td>
<td>139</td>
<td>62</td>
<td>47</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNFLOWER BLUE</td>
<td>100</td>
<td>149</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>CORNFLOWER BLUE1</td>
<td>100</td>
<td>149</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>CORNSILK</td>
<td>255</td>
<td>248</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>CORNSILK1</td>
<td>255</td>
<td>248</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>CORNSILK2</td>
<td>238</td>
<td>232</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>CORNSILK3</td>
<td>205</td>
<td>200</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>CORNSILK4</td>
<td>139</td>
<td>136</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>CYAN</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>CYAN1</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>CYAN2</td>
<td>0</td>
<td>238</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>CYAN3</td>
<td>0</td>
<td>205</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>CYAN4</td>
<td>0</td>
<td>139</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARK BLUE</td>
<td>0</td>
<td>0</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARK CYAN</td>
<td>0</td>
<td>139</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARK GOLDENROD</td>
<td>184</td>
<td>134</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>DARK GRAY</td>
<td>169</td>
<td>169</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>DARK GREEN</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARK GREY</td>
<td>169</td>
<td>169</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>DARK KHAKI</td>
<td>189</td>
<td>183</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>DARK MAGENTA</td>
<td>139</td>
<td>0</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARK OLIVE GREEN</td>
<td>85</td>
<td>107</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>DARK ORANGE</td>
<td>255</td>
<td>140</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARK ORCHID</td>
<td>153</td>
<td>50</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>DARK RED</td>
<td>139</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARK SALMON</td>
<td>233</td>
<td>150</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>DARK SEA GREEN</td>
<td>143</td>
<td>188</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>DARK SLATE BLUE</td>
<td>72</td>
<td>61</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARK SLATE GRAY</td>
<td>47</td>
<td>79</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>DARK SLATE GREY</td>
<td>47</td>
<td>79</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>DARK TURQUOISE</td>
<td>0</td>
<td>206</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>DARK VIOLET</td>
<td>148</td>
<td>0</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>DARKBLUE</td>
<td>0</td>
<td>0</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARKCYAN</td>
<td>0</td>
<td>139</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARKGOLDENROD</td>
<td>184</td>
<td>134</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>DARKGOLDENROD1</td>
<td>255</td>
<td>185</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>DARKGOLDENROD2</td>
<td>238</td>
<td>173</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>DARKGOLDENROD3</td>
<td>205</td>
<td>149</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>DARKGOLDENROD4</td>
<td>139</td>
<td>101</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>DARKGRAY</td>
<td>169</td>
<td>169</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>DARKGREEN</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARKGREY</td>
<td>169</td>
<td>169</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>DARKKHAKI</td>
<td>189</td>
<td>183</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>DARKMAGENTA</td>
<td>139</td>
<td>0</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
<td>Example</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>DARKOLIVEGREEN</td>
<td>85</td>
<td>107</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>DARKOLIVEGREEN1</td>
<td>202</td>
<td>255</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>DARKOLIVEGREEN2</td>
<td>188</td>
<td>238</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>DARKOLIVEGREEN3</td>
<td>162</td>
<td>205</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>DARKOLIVEGREEN4</td>
<td>110</td>
<td>139</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>DARKORANGE</td>
<td>255</td>
<td>140</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARKORANGE1</td>
<td>255</td>
<td>127</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARKORANGE2</td>
<td>238</td>
<td>118</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARKORANGE3</td>
<td>205</td>
<td>102</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARKORANGE4</td>
<td>139</td>
<td>69</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DARKORCHID</td>
<td>153</td>
<td>50</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>DARKORCHID1</td>
<td>191</td>
<td>62</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>DARKORCHID2</td>
<td>178</td>
<td>58</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>DARKORCHID3</td>
<td>154</td>
<td>50</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>DARKORCHID4</td>
<td>104</td>
<td>34</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>DARKRED</td>
<td>139</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Darksalmon</td>
<td>233</td>
<td>150</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>Darkseagreen</td>
<td>143</td>
<td>188</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>DARKSEAGREEN1</td>
<td>193</td>
<td>255</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>DARKSEAGREEN2</td>
<td>180</td>
<td>238</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>DARKSEAGREEN3</td>
<td>155</td>
<td>205</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>DARKSEAGREEN4</td>
<td>105</td>
<td>139</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>Darkslateblue</td>
<td>72</td>
<td>61</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Darkslategray</td>
<td>47</td>
<td>79</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>DARKSLATEGRAY1</td>
<td>151</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>DARKSLATEGRAY2</td>
<td>141</td>
<td>238</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>DARKSLATEGRAY3</td>
<td>121</td>
<td>205</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>DARKSLATEGRAY4</td>
<td>82</td>
<td>139</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Darkslategrey</td>
<td>47</td>
<td>79</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>Darkturquoise</td>
<td>0</td>
<td>206</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>Darkviolet</td>
<td>148</td>
<td>0</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>Deep pink</td>
<td>255</td>
<td>20</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>Deep sky blue</td>
<td>0</td>
<td>191</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Deep pink</td>
<td>255</td>
<td>20</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>Deep pink1</td>
<td>255</td>
<td>20</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>Deep pink2</td>
<td>238</td>
<td>18</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>Deep pink3</td>
<td>205</td>
<td>16</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Deep pink4</td>
<td>139</td>
<td>10</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Deepskyblue</td>
<td>0</td>
<td>191</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Deepskyblue1</td>
<td>0</td>
<td>191</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Deepskyblue2</td>
<td>0</td>
<td>178</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>Deepskyblue3</td>
<td>0</td>
<td>154</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>Deepskyblue4</td>
<td>0</td>
<td>104</td>
<td>139</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIM GRAY</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>DIM GREY</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>DIMGRAY</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>DIMGREY</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>DODGER BLUE</td>
<td>30</td>
<td>144</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>DODGERBLUE</td>
<td>30</td>
<td>144</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>DODGERBLUE1</td>
<td>30</td>
<td>144</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>DODGERBLUE2</td>
<td>28</td>
<td>134</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>DODGERBLUE3</td>
<td>24</td>
<td>116</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>DODGERBLUE4</td>
<td>16</td>
<td>78</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>FIREBRICK</td>
<td>178</td>
<td>34</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>FIREBRICK1</td>
<td>255</td>
<td>48</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>FIREBRICK2</td>
<td>238</td>
<td>44</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>FIREBRICK3</td>
<td>205</td>
<td>38</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>FIREBRICK4</td>
<td>139</td>
<td>26</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>FLORAL WHITE</td>
<td>255</td>
<td>250</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>FLORALWHITE</td>
<td>255</td>
<td>250</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>FOREST GREEN</td>
<td>34</td>
<td>139</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>FORESTGREEN</td>
<td>34</td>
<td>139</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>GAINSBORO</td>
<td>220</td>
<td>220</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>GHOST WHITE</td>
<td>248</td>
<td>248</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>GHOSTWHITE</td>
<td>248</td>
<td>248</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>GOLD</td>
<td>255</td>
<td>215</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GOLD1</td>
<td>255</td>
<td>215</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GOLD2</td>
<td>238</td>
<td>201</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GOLD3</td>
<td>205</td>
<td>173</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GOLD4</td>
<td>139</td>
<td>117</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GOLDENROD</td>
<td>218</td>
<td>165</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>GOLDENROD1</td>
<td>255</td>
<td>193</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>GOLDENROD2</td>
<td>238</td>
<td>180</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>GOLDENROD3</td>
<td>205</td>
<td>155</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>GOLDENROD4</td>
<td>139</td>
<td>105</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>GRAY</td>
<td>190</td>
<td>190</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>GRAY0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GRAY1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>GRAY2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>GRAY3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>GRAY4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>GRAY5</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>GRAY6</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>GRAY7</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>GRAY8</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>GRAY9</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 9: Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAY10</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>GRAY11</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>GRAY12</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>GRAY13</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>GRAY14</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>GRAY15</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>GRAY16</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>GRAY17</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>GRAY18</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>GRAY19</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>GRAY20</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>GRAY21</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>GRAY22</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>GRAY23</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>GRAY24</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>GRAY25</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>GRAY26</td>
<td>66</td>
<td>66</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>GRAY27</td>
<td>69</td>
<td>69</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>GRAY28</td>
<td>71</td>
<td>71</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>GRAY29</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>GRAY30</td>
<td>77</td>
<td>77</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>GRAY31</td>
<td>79</td>
<td>79</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>GRAY32</td>
<td>82</td>
<td>82</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>GRAY33</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>GRAY34</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>GRAY35</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>GRAY36</td>
<td>92</td>
<td>92</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>GRAY37</td>
<td>94</td>
<td>94</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>GRAY38</td>
<td>97</td>
<td>97</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>GRAY39</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>GRAY40</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>GRAY41</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>GRAY42</td>
<td>107</td>
<td>107</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>GRAY43</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>GRAY44</td>
<td>112</td>
<td>112</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>GRAY45</td>
<td>115</td>
<td>115</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>GRAY46</td>
<td>117</td>
<td>117</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>GRAY47</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>GRAY48</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>GRAY49</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>GRAY50</td>
<td>127</td>
<td>127</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>GRAY51</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>GRAY52</td>
<td>133</td>
<td>133</td>
<td>133</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAY53</td>
<td>135</td>
<td>135</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>GRAY54</td>
<td>138</td>
<td>138</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>GRAY55</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>GRAY56</td>
<td>143</td>
<td>143</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>GRAY57</td>
<td>145</td>
<td>145</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>GRAY58</td>
<td>148</td>
<td>148</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>GRAY59</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>GRAY60</td>
<td>153</td>
<td>153</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>GRAY61</td>
<td>156</td>
<td>156</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>GRAY62</td>
<td>158</td>
<td>158</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>GRAY63</td>
<td>161</td>
<td>161</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>GRAY64</td>
<td>163</td>
<td>163</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>GRAY65</td>
<td>166</td>
<td>166</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>GRAY66</td>
<td>168</td>
<td>168</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>GRAY67</td>
<td>171</td>
<td>171</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>GRAY68</td>
<td>173</td>
<td>173</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>GRAY69</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>GRAY70</td>
<td>179</td>
<td>179</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>GRAY71</td>
<td>181</td>
<td>181</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>GRAY72</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>GRAY73</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td>GRAY74</td>
<td>189</td>
<td>189</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>GRAY75</td>
<td>191</td>
<td>191</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>GRAY76</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>GRAY77</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>GRAY78</td>
<td>199</td>
<td>199</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>GRAY79</td>
<td>201</td>
<td>201</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>GRAY80</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>GRAY81</td>
<td>207</td>
<td>207</td>
<td>207</td>
<td></td>
</tr>
<tr>
<td>GRAY82</td>
<td>209</td>
<td>209</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>GRAY83</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>GRAY84</td>
<td>214</td>
<td>214</td>
<td>214</td>
<td></td>
</tr>
<tr>
<td>GRAY85</td>
<td>217</td>
<td>217</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>GRAY86</td>
<td>219</td>
<td>219</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>GRAY87</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>GRAY88</td>
<td>224</td>
<td>224</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>GRAY89</td>
<td>227</td>
<td>227</td>
<td>227</td>
<td></td>
</tr>
<tr>
<td>GRAY90</td>
<td>229</td>
<td>229</td>
<td>229</td>
<td></td>
</tr>
<tr>
<td>GRAY91</td>
<td>232</td>
<td>232</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td>GRAY92</td>
<td>235</td>
<td>235</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>GRAY93</td>
<td>237</td>
<td>237</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>GRAY94</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>GRAY95</td>
<td>242</td>
<td>242</td>
<td>242</td>
<td></td>
</tr>
</tbody>
</table>
## Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAY96</td>
<td>245</td>
<td>245</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>GRAY97</td>
<td>247</td>
<td>247</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>GRAY98</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>GRAY99</td>
<td>252</td>
<td>252</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>GRAY100</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>GREEN</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GREEN YELLOW</td>
<td>173</td>
<td>255</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>GREEN1</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GREEN2</td>
<td>0</td>
<td>238</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GREEN3</td>
<td>0</td>
<td>205</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GREEN4</td>
<td>0</td>
<td>139</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GREENYELLOW</td>
<td>173</td>
<td>255</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>GREY</td>
<td>190</td>
<td>190</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>GREY0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GREY1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>GREY2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>GREY3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>GREY4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>GREY5</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>GREY6</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>GREY7</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>GREY8</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>GREY9</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>GREY10</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>GREY11</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>GREY12</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>GREY13</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>GREY14</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>GREY15</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>GREY16</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>GREY17</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>GREY18</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>GREY19</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>GREY20</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>GREY21</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>GREY22</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>GREY23</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>GREY24</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>GREY25</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>GREY26</td>
<td>66</td>
<td>66</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>GREY27</td>
<td>69</td>
<td>69</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>GREY28</td>
<td>71</td>
<td>71</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>GREY29</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td></td>
</tr>
</tbody>
</table>
## Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREY30</td>
<td>77</td>
<td>77</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>GREY31</td>
<td>79</td>
<td>79</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>GREY32</td>
<td>82</td>
<td>82</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>GREY33</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>GREY34</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>GREY35</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>GREY36</td>
<td>92</td>
<td>92</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>GREY37</td>
<td>94</td>
<td>94</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>GREY38</td>
<td>97</td>
<td>97</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>GREY39</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>GREY40</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>GREY41</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>GREY42</td>
<td>107</td>
<td>107</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>GREY43</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>GREY44</td>
<td>112</td>
<td>112</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>GREY45</td>
<td>115</td>
<td>115</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>GREY46</td>
<td>117</td>
<td>117</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>GREY47</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>GREY48</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>GREY49</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>GREY50</td>
<td>127</td>
<td>127</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>GREY51</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>GREY52</td>
<td>133</td>
<td>133</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>GREY53</td>
<td>135</td>
<td>135</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>GREY54</td>
<td>138</td>
<td>138</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>GREY55</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>GREY56</td>
<td>143</td>
<td>143</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>GREY57</td>
<td>145</td>
<td>145</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>GREY58</td>
<td>148</td>
<td>148</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>GREY59</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>GREY60</td>
<td>153</td>
<td>153</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>GREY61</td>
<td>156</td>
<td>156</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>GREY62</td>
<td>158</td>
<td>158</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>GREY63</td>
<td>161</td>
<td>161</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>GREY64</td>
<td>163</td>
<td>163</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>GREY65</td>
<td>166</td>
<td>166</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>GREY66</td>
<td>168</td>
<td>168</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>GREY67</td>
<td>171</td>
<td>171</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>GREY68</td>
<td>173</td>
<td>173</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>GREY69</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>GREY70</td>
<td>179</td>
<td>179</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>GREY71</td>
<td>181</td>
<td>181</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>GREY72</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREY73</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td>GREY74</td>
<td>189</td>
<td>189</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>GREY75</td>
<td>191</td>
<td>191</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>GREY76</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>GREY77</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>GREY78</td>
<td>199</td>
<td>199</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>GREY79</td>
<td>201</td>
<td>201</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>GREY80</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>GREY81</td>
<td>207</td>
<td>207</td>
<td>207</td>
<td></td>
</tr>
<tr>
<td>GREY82</td>
<td>209</td>
<td>209</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>GREY83</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>GREY84</td>
<td>214</td>
<td>214</td>
<td>214</td>
<td></td>
</tr>
<tr>
<td>GREY85</td>
<td>217</td>
<td>217</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>GREY86</td>
<td>219</td>
<td>219</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>GREY87</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>GREY88</td>
<td>224</td>
<td>224</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>GREY89</td>
<td>227</td>
<td>227</td>
<td>227</td>
<td></td>
</tr>
<tr>
<td>GREY90</td>
<td>229</td>
<td>229</td>
<td>229</td>
<td></td>
</tr>
<tr>
<td>GREY91</td>
<td>232</td>
<td>232</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td>GREY92</td>
<td>235</td>
<td>235</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>GREY93</td>
<td>237</td>
<td>237</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>GREY94</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>GREY95</td>
<td>242</td>
<td>242</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>GREY96</td>
<td>245</td>
<td>245</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>GREY97</td>
<td>247</td>
<td>247</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>GREY98</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>GREY99</td>
<td>252</td>
<td>252</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>GREY100</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>HONEYDEW</td>
<td>240</td>
<td>255</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>HONEYDEW1</td>
<td>240</td>
<td>255</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>HONEYDEW2</td>
<td>224</td>
<td>238</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>HONEYDEW3</td>
<td>193</td>
<td>205</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>HONEYDEW4</td>
<td>131</td>
<td>139</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>HOT PINK</td>
<td>255</td>
<td>105</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>HOTPINK</td>
<td>255</td>
<td>105</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>HOTPINK1</td>
<td>255</td>
<td>110</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>HOTPINK2</td>
<td>238</td>
<td>106</td>
<td>167</td>
<td></td>
</tr>
<tr>
<td>HOTPINK3</td>
<td>205</td>
<td>96</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>HOTPINK4</td>
<td>139</td>
<td>58</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>INDIAN RED</td>
<td>205</td>
<td>92</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>INDIANRED</td>
<td>205</td>
<td>92</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>INDIANRED1</td>
<td>255</td>
<td>106</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>INDIANRED2</td>
<td>238</td>
<td>99</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
<td>Example</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>INDIANRED3</td>
<td>205</td>
<td>85</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>INDIANRED4</td>
<td>139</td>
<td>58</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>IVORY</td>
<td>255</td>
<td>255</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>IVORY1</td>
<td>255</td>
<td>255</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>IVORY2</td>
<td>238</td>
<td>238</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>IVORY3</td>
<td>205</td>
<td>205</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>IVORY4</td>
<td>139</td>
<td>139</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>KHAKI</td>
<td>240</td>
<td>230</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>KHAKI1</td>
<td>255</td>
<td>246</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>KHAKI2</td>
<td>238</td>
<td>230</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>KHAKI3</td>
<td>205</td>
<td>198</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>KHAKI4</td>
<td>139</td>
<td>134</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>LAVENDER</td>
<td>230</td>
<td>230</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>LAVENDER BLUSH</td>
<td>255</td>
<td>240</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>LAVENDERBLUSH</td>
<td>255</td>
<td>240</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>LAVENDERBLUSH1</td>
<td>255</td>
<td>240</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>LAVENDERBLUSH2</td>
<td>238</td>
<td>224</td>
<td>229</td>
<td></td>
</tr>
<tr>
<td>LAVENDERBLUSH3</td>
<td>205</td>
<td>193</td>
<td>197</td>
<td></td>
</tr>
<tr>
<td>LAVENDERBLUSH4</td>
<td>139</td>
<td>131</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>LAWN GREEN</td>
<td>124</td>
<td>252</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>LAWN GREEN</td>
<td>124</td>
<td>252</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>LEMON CHIFFON</td>
<td>255</td>
<td>250</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>LEMONCHIFFON</td>
<td>255</td>
<td>250</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>LEMONCHIFFON1</td>
<td>255</td>
<td>250</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>LEMONCHIFFON2</td>
<td>238</td>
<td>233</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>LEMONCHIFFON3</td>
<td>205</td>
<td>201</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>LEMONCHIFFON4</td>
<td>139</td>
<td>137</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>LIGHT BLUE</td>
<td>173</td>
<td>216</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>LIGHT CORAL</td>
<td>240</td>
<td>128</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>LIGHT CYAN</td>
<td>224</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>LIGHT GOLDENROD YELLO</td>
<td>250</td>
<td>250</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>LIGHT GOLDENROD</td>
<td>238</td>
<td>221</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>LIGHT GRAY</td>
<td>211</td>
<td>211</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>LIGHT GREEN</td>
<td>144</td>
<td>238</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>LIGHT GREY</td>
<td>211</td>
<td>211</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>LIGHT PINK</td>
<td>255</td>
<td>182</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>LIGHT SALMON</td>
<td>255</td>
<td>160</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>LIGHT SEA GREEN</td>
<td>32</td>
<td>178</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>LIGHT SKY BLUE</td>
<td>135</td>
<td>206</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>LIGHT SLATE BLUE</td>
<td>132</td>
<td>112</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>LIGHT SLATE GRAY</td>
<td>119</td>
<td>136</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>LIGHT SLATE GREY</td>
<td>119</td>
<td>136</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>LIGHT STEEL BLUE</td>
<td>176</td>
<td>196</td>
<td>222</td>
<td></td>
</tr>
</tbody>
</table>
## Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIGHT YELLOW</td>
<td>255</td>
<td>255</td>
<td>224</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTBLUE</td>
<td>173</td>
<td>216</td>
<td>230</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTBLUE1</td>
<td>191</td>
<td>239</td>
<td>255</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTBLUE2</td>
<td>178</td>
<td>223</td>
<td>238</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTBLUE3</td>
<td>154</td>
<td>192</td>
<td>205</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTBLUE4</td>
<td>104</td>
<td>131</td>
<td>139</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTCORAL</td>
<td>240</td>
<td>128</td>
<td>128</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTCYAN</td>
<td>224</td>
<td>255</td>
<td>255</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTCYAN1</td>
<td>224</td>
<td>255</td>
<td>255</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTCYAN2</td>
<td>209</td>
<td>238</td>
<td>238</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTCYAN3</td>
<td>180</td>
<td>205</td>
<td>205</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTCYAN4</td>
<td>122</td>
<td>139</td>
<td>139</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGOLDENROD</td>
<td>238</td>
<td>221</td>
<td>130</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGOLDENROD1</td>
<td>255</td>
<td>236</td>
<td>139</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGOLDENROD2</td>
<td>238</td>
<td>220</td>
<td>130</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGOLDENROD3</td>
<td>205</td>
<td>190</td>
<td>112</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGOLDENROD4</td>
<td>139</td>
<td>129</td>
<td>76</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGOLDENRODYELLOW</td>
<td>250</td>
<td>250</td>
<td>210</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGRAY</td>
<td>211</td>
<td>211</td>
<td>211</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGREEN</td>
<td>144</td>
<td>238</td>
<td>144</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTGREY</td>
<td>211</td>
<td>211</td>
<td>211</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTPINK</td>
<td>255</td>
<td>182</td>
<td>193</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTPINK1</td>
<td>255</td>
<td>174</td>
<td>185</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTPINK2</td>
<td>238</td>
<td>162</td>
<td>173</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTPINK3</td>
<td>205</td>
<td>140</td>
<td>149</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTPINK4</td>
<td>139</td>
<td>95</td>
<td>101</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSALMON</td>
<td>255</td>
<td>160</td>
<td>122</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSALMON1</td>
<td>255</td>
<td>160</td>
<td>122</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSALMON2</td>
<td>238</td>
<td>149</td>
<td>114</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSALMON3</td>
<td>205</td>
<td>129</td>
<td>98</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSALMON4</td>
<td>139</td>
<td>87</td>
<td>66</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSEAGREEN</td>
<td>32</td>
<td>178</td>
<td>170</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSKYBLUE</td>
<td>135</td>
<td>206</td>
<td>250</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSKYBLUE1</td>
<td>176</td>
<td>226</td>
<td>255</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSKYBLUE2</td>
<td>164</td>
<td>211</td>
<td>238</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSKYBLUE3</td>
<td>141</td>
<td>182</td>
<td>205</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSKYBLUE4</td>
<td>96</td>
<td>123</td>
<td>139</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSLATEBLUE</td>
<td>132</td>
<td>112</td>
<td>255</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSLATEGRAY</td>
<td>119</td>
<td>136</td>
<td>153</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSLATEGREY</td>
<td>119</td>
<td>136</td>
<td>153</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSTEELBLUE</td>
<td>176</td>
<td>196</td>
<td>222</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSTEELBLUE1</td>
<td>202</td>
<td>225</td>
<td>255</td>
<td><img src="#" alt="Example" /></td>
</tr>
<tr>
<td>LIGHTSTEELBLUE2</td>
<td>188</td>
<td>210</td>
<td>238</td>
<td><img src="#" alt="Example" /></td>
</tr>
</tbody>
</table>
### Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIGHTSTEELBLUE3</td>
<td>162</td>
<td>181</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>LIGHTSTEELBLUE4</td>
<td>110</td>
<td>123</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>LIGHTYELLOW</td>
<td>255</td>
<td>255</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>LIGHTYELLOW1</td>
<td>255</td>
<td>255</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>LIGHTYELLOW2</td>
<td>238</td>
<td>238</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>LIGHTYELLOW3</td>
<td>205</td>
<td>205</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>LIGHTYELLOW4</td>
<td>139</td>
<td>139</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>LIME GREEN</td>
<td>50</td>
<td>205</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>LIMEGREEN</td>
<td>50</td>
<td>205</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>LINEN</td>
<td>230</td>
<td>240</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>MAGENTA</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>MAGENTA1</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>MAGENTA2</td>
<td>238</td>
<td>0</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>MAGENTA3</td>
<td>205</td>
<td>0</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>MAGENTA4</td>
<td>139</td>
<td>0</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>MAROON</td>
<td>176</td>
<td>48</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>MAROON1</td>
<td>255</td>
<td>52</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>MAROON2</td>
<td>238</td>
<td>48</td>
<td>167</td>
<td></td>
</tr>
<tr>
<td>MAROON3</td>
<td>205</td>
<td>41</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>MAROON4</td>
<td>139</td>
<td>28</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>MEDIUM AQUAMARINE</td>
<td>102</td>
<td>205</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>MEDIUM BLUE</td>
<td>0</td>
<td>0</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>MEDIUM ORCHID</td>
<td>186</td>
<td>85</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>MEDIUM PURPLE</td>
<td>147</td>
<td>112</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>MEDIUM SEA GREEN</td>
<td>60</td>
<td>179</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>MEDIUM SLATE BLUE</td>
<td>123</td>
<td>104</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>MEDIUM SPRING GREEN</td>
<td>0</td>
<td>250</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>MEDIUM TURQUOISE</td>
<td>72</td>
<td>209</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>MEDIUM VIOLET RED</td>
<td>199</td>
<td>21</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>MEDIUMAQUAMARINE</td>
<td>102</td>
<td>205</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>MEDIUMBLUE</td>
<td>0</td>
<td>0</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>MEDIUMMORCHID</td>
<td>186</td>
<td>85</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>MEDIUMMORCHID1</td>
<td>224</td>
<td>102</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>MEDIUMMORCHID2</td>
<td>209</td>
<td>95</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>MEDIUMMORCHID3</td>
<td>180</td>
<td>82</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>MEDIUMMORCHID4</td>
<td>122</td>
<td>55</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>MEDIUMPURPLE</td>
<td>147</td>
<td>112</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>MEDIUMPURPLE1</td>
<td>171</td>
<td>130</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>MEDIUMPURPLE2</td>
<td>159</td>
<td>121</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>MEDIUMPURPLE3</td>
<td>137</td>
<td>104</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>MEDIUMPURPLE4</td>
<td>93</td>
<td>71</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>MEDIUMSEAGREEN</td>
<td>60</td>
<td>179</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>MEDIUMSLATEBLUE</td>
<td>123</td>
<td>104</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
<td>Example</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>MEDIUMSPRINGGREEN</td>
<td>0</td>
<td>250</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>MEDIUMLTURQUOISE</td>
<td>72</td>
<td>209</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>MEDIUMVIOLETRED</td>
<td>199</td>
<td>21</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>MIDNIGHT BLUE</td>
<td>25</td>
<td>25</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>MIDNIGHTBLUE</td>
<td>25</td>
<td>25</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>MINT CREAM</td>
<td>245</td>
<td>255</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>MINTCREAM</td>
<td>245</td>
<td>255</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>MISTY ROSE</td>
<td>255</td>
<td>228</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>MISTYROSE</td>
<td>255</td>
<td>228</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>MISTYROSE1</td>
<td>255</td>
<td>228</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>MISTYROSE2</td>
<td>238</td>
<td>213</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>MISTYROSE3</td>
<td>205</td>
<td>183</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>MISTYROSE4</td>
<td>139</td>
<td>125</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>MOCCASIN</td>
<td>255</td>
<td>228</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>NAVAJO WHITE</td>
<td>255</td>
<td>222</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>NAVAJOWHITE</td>
<td>255</td>
<td>222</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>NAVAJOWHITE1</td>
<td>255</td>
<td>222</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>NAVAJOWHITE2</td>
<td>238</td>
<td>207</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>NAVAJOWHITE3</td>
<td>205</td>
<td>179</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>NAVAJOWHITE4</td>
<td>139</td>
<td>121</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>NAVY</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>NAVY BLUE</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>NAVYBLUE</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>OLD LACE</td>
<td>253</td>
<td>245</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>OLDLACE</td>
<td>253</td>
<td>245</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>OLIVE DRAB</td>
<td>107</td>
<td>142</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>OLIVEDRAB</td>
<td>107</td>
<td>142</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>OLIVEDRAB1</td>
<td>192</td>
<td>255</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>OLIVEDRAB2</td>
<td>179</td>
<td>238</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>OLIVEDRAB3</td>
<td>154</td>
<td>205</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>OLIVEDRAB4</td>
<td>105</td>
<td>139</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>ORANGE</td>
<td>255</td>
<td>165</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGE RED</td>
<td>255</td>
<td>69</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGE1</td>
<td>255</td>
<td>165</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGE2</td>
<td>238</td>
<td>154</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGE3</td>
<td>205</td>
<td>133</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGE4</td>
<td>139</td>
<td>90</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGERED</td>
<td>255</td>
<td>69</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGERED1</td>
<td>255</td>
<td>69</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGERED2</td>
<td>238</td>
<td>64</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGERED3</td>
<td>205</td>
<td>55</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORANGERED4</td>
<td>139</td>
<td>37</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ORCHID</td>
<td>218</td>
<td>112</td>
<td>214</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORCHID1</td>
<td>255</td>
<td>131</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>ORCHID2</td>
<td>238</td>
<td>122</td>
<td>233</td>
<td></td>
</tr>
<tr>
<td>ORCHID3</td>
<td>205</td>
<td>105</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>ORCHID4</td>
<td>139</td>
<td>71</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>PALE GOLDENROD</td>
<td>238</td>
<td>232</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>PALE GREEN</td>
<td>152</td>
<td>251</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>PALE TURQUOISE</td>
<td>175</td>
<td>238</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>PALE VIOLET RED</td>
<td>219</td>
<td>112</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>PALEGOLDENROD</td>
<td>238</td>
<td>232</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>PALEGREEN</td>
<td>152</td>
<td>251</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>PALEGREEN1</td>
<td>154</td>
<td>255</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>PALEGREEN2</td>
<td>144</td>
<td>238</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>PALEGREEN3</td>
<td>124</td>
<td>205</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>PALEGREEN4</td>
<td>84</td>
<td>139</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>PALETURQUOISE</td>
<td>175</td>
<td>238</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>PALETURQUOISE1</td>
<td>187</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>PALETURQUOISE2</td>
<td>174</td>
<td>238</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>PALETURQUOISE3</td>
<td>150</td>
<td>205</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>PALETURQUOISE4</td>
<td>102</td>
<td>139</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>PALEVIOLETRED</td>
<td>219</td>
<td>112</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>PALEVIOLETRED1</td>
<td>255</td>
<td>130</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>PALEVIOLETRED2</td>
<td>238</td>
<td>121</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>PALEVIOLETRED3</td>
<td>205</td>
<td>104</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>PALEVIOLETRED4</td>
<td>139</td>
<td>71</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>PAPAYA WHIP</td>
<td>255</td>
<td>239</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td>PAPAYAWHIP</td>
<td>255</td>
<td>239</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td>PEACH PUFF</td>
<td>255</td>
<td>218</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>PEACHPUFF</td>
<td>255</td>
<td>218</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>PEACHPUFF1</td>
<td>255</td>
<td>218</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>PEACHPUFF2</td>
<td>238</td>
<td>203</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>PEACHPUFF3</td>
<td>205</td>
<td>175</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>PEACHPUFF4</td>
<td>139</td>
<td>119</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>PERU</td>
<td>205</td>
<td>133</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>PINK</td>
<td>255</td>
<td>192</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>PINK1</td>
<td>255</td>
<td>181</td>
<td>197</td>
<td></td>
</tr>
<tr>
<td>PINK2</td>
<td>238</td>
<td>169</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>PINK3</td>
<td>205</td>
<td>145</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>PINK4</td>
<td>139</td>
<td>99</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>PLUM</td>
<td>221</td>
<td>160</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>PLUM1</td>
<td>255</td>
<td>187</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>PLUM2</td>
<td>238</td>
<td>174</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>PLUM3</td>
<td>205</td>
<td>150</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>PLUM4</td>
<td>139</td>
<td>102</td>
<td>139</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWDER BLUE</td>
<td>176</td>
<td>224</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>POWDERBLUE</td>
<td>176</td>
<td>224</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>PURPLE</td>
<td>160</td>
<td>32</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>PURPLE1</td>
<td>155</td>
<td>48</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>PURPLE2</td>
<td>145</td>
<td>44</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>PURPLE3</td>
<td>125</td>
<td>38</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>PURPLE4</td>
<td>85</td>
<td>26</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>RED</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RED1</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RED2</td>
<td>238</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RED3</td>
<td>205</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RED4</td>
<td>139</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ROSY BROWN</td>
<td>188</td>
<td>143</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>ROSYBROWN</td>
<td>188</td>
<td>143</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>ROSYBROWN1</td>
<td>255</td>
<td>193</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>ROSYBROWN2</td>
<td>238</td>
<td>180</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>ROSYBROWN3</td>
<td>205</td>
<td>155</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>ROSYBROWN4</td>
<td>139</td>
<td>105</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>ROYAL BLUE</td>
<td>65</td>
<td>105</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>ROYALBLUE</td>
<td>65</td>
<td>105</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>ROYALBLUE1</td>
<td>72</td>
<td>118</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>ROYALBLUE2</td>
<td>67</td>
<td>110</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>ROYALBLUE3</td>
<td>58</td>
<td>95</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>ROYALBLUE4</td>
<td>39</td>
<td>64</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>SADDLE BROWN</td>
<td>139</td>
<td>69</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>SADDLEBROWN</td>
<td>139</td>
<td>69</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>SALMON</td>
<td>250</td>
<td>128</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>SALMON1</td>
<td>255</td>
<td>140</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>SALMON2</td>
<td>238</td>
<td>130</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>SALMON3</td>
<td>205</td>
<td>112</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>SALMON4</td>
<td>139</td>
<td>76</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>SANDY BROWN</td>
<td>244</td>
<td>164</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>SANDYBROWN</td>
<td>244</td>
<td>164</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>SEA GREEN</td>
<td>46</td>
<td>139</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>SEAGREEN</td>
<td>46</td>
<td>139</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>SEAGREEN1</td>
<td>84</td>
<td>255</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>SEAGREEN2</td>
<td>78</td>
<td>238</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>SEAGREEN3</td>
<td>67</td>
<td>205</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>SEAGREEN4</td>
<td>46</td>
<td>139</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>SEASHELL</td>
<td>255</td>
<td>245</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>SEASHELL1</td>
<td>255</td>
<td>245</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>SEASHELL2</td>
<td>238</td>
<td>229</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>SEASHELL3</td>
<td>205</td>
<td>197</td>
<td>191</td>
<td></td>
</tr>
</tbody>
</table>
## Chapter 9. Color Name Reference

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEASHELL4</td>
<td>139</td>
<td>134</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>SIENNA</td>
<td>160</td>
<td>82</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>SIENNA1</td>
<td>255</td>
<td>130</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>SIENNA2</td>
<td>238</td>
<td>121</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>SIENNA3</td>
<td>205</td>
<td>104</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>SIENNA4</td>
<td>139</td>
<td>71</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>SKY BLUE</td>
<td>135</td>
<td>206</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>SKYBLUE</td>
<td>135</td>
<td>206</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>SKYBLUE1</td>
<td>135</td>
<td>206</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>SKYBLUE2</td>
<td>126</td>
<td>192</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>SKYBLUE3</td>
<td>108</td>
<td>166</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>SKYBLUE4</td>
<td>74</td>
<td>112</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>SLATE BLUE</td>
<td>106</td>
<td>90</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>SLATE GRAY</td>
<td>112</td>
<td>128</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>SLATE GREY</td>
<td>112</td>
<td>128</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>SLATEBLUE</td>
<td>106</td>
<td>90</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>SLATEBLUE1</td>
<td>131</td>
<td>111</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>SLATEBLUE2</td>
<td>122</td>
<td>103</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>SLATEBLUE3</td>
<td>105</td>
<td>89</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>SLATEBLUE4</td>
<td>71</td>
<td>60</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>SLATEGRAY</td>
<td>112</td>
<td>128</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>SLATEGRAY1</td>
<td>198</td>
<td>226</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>SLATEGRAY2</td>
<td>185</td>
<td>211</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>SLATEGRAY3</td>
<td>139</td>
<td>182</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>SLATEGRAY4</td>
<td>108</td>
<td>123</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>SLATEGREY</td>
<td>112</td>
<td>128</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>SNOW</td>
<td>255</td>
<td>250</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>SNOW1</td>
<td>255</td>
<td>250</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>SNOW2</td>
<td>238</td>
<td>233</td>
<td>233</td>
<td></td>
</tr>
<tr>
<td>SNOW3</td>
<td>205</td>
<td>201</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>SNOW4</td>
<td>139</td>
<td>137</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>SPRING GREEN</td>
<td>0</td>
<td>255</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>SPRINGGREEN</td>
<td>0</td>
<td>255</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>SPRINGGREEN1</td>
<td>0</td>
<td>255</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>SPRINGGREEN2</td>
<td>0</td>
<td>238</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>SPRINGGREEN3</td>
<td>0</td>
<td>205</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>SPRINGGREEN4</td>
<td>0</td>
<td>139</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>STEEL BLUE</td>
<td>70</td>
<td>130</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>STEELBLUE</td>
<td>70</td>
<td>130</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>STEELBLUE1</td>
<td>99</td>
<td>184</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>STEELBLUE2</td>
<td>92</td>
<td>172</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>STEELBLUE3</td>
<td>79</td>
<td>148</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>STEELBLUE4</td>
<td>54</td>
<td>100</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
<td>Example</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>TAN</td>
<td>210</td>
<td>180</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>TAN1</td>
<td>255</td>
<td>165</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>TAN2</td>
<td>238</td>
<td>154</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>TAN3</td>
<td>205</td>
<td>133</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>TAN4</td>
<td>139</td>
<td>90</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>THISTLE</td>
<td>216</td>
<td>191</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>THISTLE1</td>
<td>255</td>
<td>225</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>THISTLE2</td>
<td>238</td>
<td>210</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>THISTLE3</td>
<td>205</td>
<td>181</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>THISTLE4</td>
<td>139</td>
<td>123</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>TOMATO</td>
<td>255</td>
<td>99</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>TOMATO1</td>
<td>255</td>
<td>99</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>TOMATO2</td>
<td>238</td>
<td>92</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>TOMATO3</td>
<td>205</td>
<td>79</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>TOMATO4</td>
<td>139</td>
<td>54</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>TURQUOISE</td>
<td>64</td>
<td>224</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td>TURQUOISE1</td>
<td>0</td>
<td>245</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>TURQUOISE2</td>
<td>0</td>
<td>229</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>TURQUOISE3</td>
<td>0</td>
<td>197</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>TURQUOISE4</td>
<td>0</td>
<td>134</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>VIOLET</td>
<td>238</td>
<td>130</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>VIOLET RED</td>
<td>208</td>
<td>32</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>VIOLET RED1</td>
<td>208</td>
<td>32</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>VIOLET RED2</td>
<td>255</td>
<td>62</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>VIOLET RED3</td>
<td>238</td>
<td>58</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>VIOLET RED4</td>
<td>205</td>
<td>50</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>WHEAT</td>
<td>245</td>
<td>222</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>WHEAT1</td>
<td>255</td>
<td>231</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td>WHEAT2</td>
<td>238</td>
<td>216</td>
<td>174</td>
<td></td>
</tr>
<tr>
<td>WHEAT3</td>
<td>205</td>
<td>186</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>WHEAT4</td>
<td>139</td>
<td>126</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>WHITE</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>WHITE SMOKE</td>
<td>245</td>
<td>245</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>WHITESMOKE</td>
<td>245</td>
<td>245</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>YELLOW</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>YELLOW GREEN</td>
<td>154</td>
<td>205</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>YELLOW1</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>YELLOW2</td>
<td>238</td>
<td>238</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>YELLOW3</td>
<td>205</td>
<td>205</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>YELLOW4</td>
<td>139</td>
<td>139</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>YELLOWGREEN</td>
<td>154</td>
<td>205</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>