Useful things about Boo

Boo.Lang.Useful - an assembly rife with things beautiful and useful yet not quite core to the Boo language. Welcome to this introductory guide, where we will plumb its murky depths in search of adventure on the high seas and delicious pirate treasures.

What's so Useful about this Useful assembly?!

Boo.Lang.Useful is meant for user contributions to Boo in Boo - sometimes you have a really good idea for a great addition to Boo, but it just isn't "core" for the language. Enter, Boo.Lang.Useful: an assembly based primarily of user contributions for stuff that is extremely helpful but not quite core. If you want to contribute, please go to JIRA and open a new issue in the "Boo.Lang.Useful" component, and attach a patch for your component. Please follow the Coding Conventions page for submissions.

Here are some of the out-standing features already available ...

Useful.IO

TextFile (class)

Summary

This class is used as a quick and dirty to enumerate through the lines in a text file as if it were any other object that implemented the IEnumerable interface. It is useful for quickly grokking information in newline-delimited (\n, or \r\n) formats. It is also useful for editing a file's contents with a short, sweet, and simple API.

The static methods ReadFile() and WriteFile() are used to respectively read the entire contents of a file to a string, and write the contents of a string to a file.

Example:

text.txt

```
Hello, world!
How are you today?

Not well, eh? Well, that's a shame.

Get well soon.
```

textReader.boo

```
""" Prints (x2):
Hello, world!
How are you today?

Not well, eh? Well, that's a shame.

Get well soon.
"""

import Useful.IO from "Boo.Lang.Useful"

for line in TextFile("text.txt"):
 print line
contents = TextFile.ReadFile("text.txt")
print contents
TextFile.WriteFile("text-newfile.txt", contents)
```

forEachFileIn (function)

Summary

eachFile accepts two parameters: a string containing a directory that will act as root, and a function that will be executed for each file in the root directory and all of the root directory's subdirectories. This function is recursive.

```
import Useful.IO
#We want to print each and every filename in this directory.
#The output will be, 'C:\My Documents\Valuable Documents\foo.txt'
forEachFileIn("""C:\My Documents\Valuable Documents\""", { fileName as string | print
fileName })
```

listFiles (function)

Summary

listFiles returns an enumerable object, a list of strings that represent filenames in the root directory and every subdirectory specified by the parameter. It is a recursive function.

Example

```
import Useful.IO
#Print every file found in this directory.
#The output will be, 'C:\My Documents\Valuable Documents\foo.txt'
files = listFiles("""C:\My Documents\Valuable Documents""")
for f in files:
    print f
```

walk (function)

Summary

walk returns an enumerable object consisting of three elements: the current directory, an array of strings representing subdirectories within the current directory, and an array of strings representing files within the current directory. It is a recursive function.

Example

```
import Useful.IO
contents = walk("""C:\My Documents\Valuable Documents""")
#Print the current directory, and the number of subdirectories and files present.
#The output is, 'The current directory is C:\My Documents\Valuable Documents\, there
are 1 subdirectories and 666 files in this directory.
for root as string, subs as (string), files as (string) in contents:
    print "The current directory is $root. There are $(len(subs)) subdirectories and
$(len(files)) files in this directory."
```

Useful.Attributes

Singleton (attribute)

Summary

The Singleton attribute automates the implementation of the Singleton design pattern. Attaching this attribute to a structure or a class generates code to protect all constructors and and implements a single property, 'Instance,' that points to an instance of your class.

If you want to initalize certain objects, create a parameter-less constructor (this will be made protected and will be called by the Singleton attribute).

For more information about the singleton design pattern, check out this article.

Example.

```
/*
Hey, hey, what do you say?
*/

import Useful.Attributes

[Singleton]
class SingletonExample:
 [property(Variable)]
 _var as string

 def constructor():
  Variable = "Hey, hey, what do you say?"


print SingletonExample.Instance.Variable
```

Once attribute

Summary

Applied to a method, the Once attribute will automatically cache the value returned by the method on it's first call. Following calls to the method will simply re-use the cached value. This is usefull for methods that do long and / or expensive processing since the processing will be done only 'Once'.
Notes:

- To avoid confusion for the user of the method, the attribute should not be applied on a method that accepts parameters, since the method will always return the same results even when called with different parameter values.
- This attribute is intended for expensive calculations, it is not recommended for short methods and properties.
- The modified method will be made thread safe through the use of a lock.

Example.

```
/*
doing expensive processing
1234
1234
*/

import Useful.Attributes

class OnceTest:

    [Once]
    def expensiveFunction():
        # Do expensive processing here
        print "doing expensive processing"
        resultOfExpensiveProcessing = 1234

        # ... and return results
        return resultOfExpensiveProcessing

onceTest = OnceTest()
print onceTest.expensiveFunction()
print onceTest.expensiveFunction()
```

Useful.Collections

Set (class)

Summary

A Set class for the math freaks. (stub, fill in later)