Builtin Functions Summary

Here is a summary of boo's built-in functions. The actual source code underlying these functions is in Boo/Lang/Builtins.cs.

array

array is a function used to create empty arrays or convert IEnumerable and ICollection objects to arrays. See Lists And Arrays for more info.

array(object as IEnumerable)

Converts an IEnumerable object to a non-specific (type of System.Object) array.

array(type as Type, collection as ICollection)

Converts any object that implements ICollection to an array of Type.

array(type as Type, enumerable as IEnumerable)

Converts any object that impelements IEnumerable to an array of Type.

array(type as Type, size as int)

Creates an empty array of the specified size.

BooVersion

BooVersion is a builtin property that returns the current version of boo that is running. It returns a System.Version class. See Getting Boo Version for more info.

enumerate

enumerate(enumerable as object) as EnumerateEnumerator

enumerate() is useful when you want to keep a running count while looping through items using a for loop:

```
mylist = ["a", "b", "c"]
for i as int, obj in enumerate(mylist):
      print i, ":", obj
```

gets

Returns a string of input that originates from Console.ReadLine() - AKA, "Standard Input."

gets()

The equivilent of Console.ReadLine(). See also prompt() below.

iterator

iterator(enumerable as object) as IEnumerable

Usually not necessary, this builtin returns any Enumerator it can find in the object passed. For loops do this for you.

join

Always returning a string, join is a function that will walk through an enumerable object and put all of those elements into one string.

join(enumerable as IEnumerable)

Joins all of the elements in enumerable into one string, using a single space (ASCII Code: 32) between elements.

join(enumerable as IEnumerable, seperator as Char)

The same as join(enumerable as IEnumerable), except that seperator defines what character seperates each element in enumerable.

map

map returns an enumerable object that applies a specific function to each element in an enumerable object.

map(enumerable as IEnumerable, function as ICallable)

Taking an enumerable object such as a list or a collection, it returns an IEnumerable object that applies "function" to each element in the array.

Examples:

```
def HardRock(item):
 return "$item totally rocks out, man!"

wyclefSongs = ("Two wrongs", "Dirty
Dancing")

x = map(wyclefSongs, HardRock)
for y in x:
 print y

//another example using a multiline
anonymous closure:
newlist = map([1,2,3,4,5,6]) def (x as int):
    return x*x*x
```

Output:

```
Two wrongs totally rocks out, man!
Dirty Dancing totally rocks out, man!
```

matrix

See Multidimensional Arrays for more info, but here is a basic example:

matrix(elementType as Type, length of first dimension as int, length of second dimension as int)

Creates a multidimensional array of type elementType with the specifications of length.

Examples

```
foo = matrix(int, 2, 2)
foo[0, 0] = 0
foo[0, 1] = 1
foo[1, 0] = 2
foo[1, 1] = 3

print join(foo) //prints "0, 1, 2, 3"
/* Looks like,
[0, 1
 2, 3]
*/
```

print

Prints an object to Standard Out. The equivilent of Console.WriteLine

print(object as Object)

The equivilent of Console.WriteLine()

prompt

Prompts the user for information.

prompt(query as string)

Prints query to standard output, then waits for a user to 'respond.' Returns a string containing what the user has typed.

range

A mysterious, somewhat exciting function that returns an enumerable object containing a list of elements such as 1 to 10 or 0 to 5 or 77 to 6.

range(max as int)

Returns an IEnumerable object that contains elements from 0 to max - 1.

```
#This prints 0 through 9:
for i in range(10):
    print i
```

range(begin as int, end as int)

Returns an IEnumerable object that contains elements from begin to end - 1.

range(begin as int, end as int, step as int)

Returns an IEnumerable object that contains all of the elements from begin to end - 1 that match the interval of step.

Example:

```
for i in range(0, 10, 2):
 print i
```

Output:

```
0
2
4
6
8
```

reversed

reversed(enumerable as object) as IEnumerable

Returns items in an enumerable in reverse order.

shell

shell is used for invoking processes and inspecting their output.

shell(filename as string, arguments as string)

Invoke an application (filename) with the arguments (arguments) specified. Returns a string containing the program's output to Standard Out (aka, the console)

shellm(filename as string, arguments as (string))

Invoke an application (filename) with an array of arguments (arguments); returns a string containing program's output.

shellp(filename as string, arguments as string)

Starts a process specified by filename with the arguments provided and returns a Process object representing the newly born process.

zip

zip returns an IEnumerable object that interleaves the two arrays.

zip(first as IEnumerable, second as IEnumerable)

zip will return an IEnumerable object, wherein each element of the IEnumerable object will be a one dimensional array containing two elements; the first element will be an element located in "firstNames" and the second will be an element located in "lastNames."

Example:

```
firstNames = ("Charles", "Joe", "P")
lastNames = ("Whittaker", "Manson", "Diddy")

x = zip(firstNames, lastNames)
for y in x:
 print join(y)
        //print y[0], y[1]
```

Output:

```
Charles Whittaker
Joe Manson
P Diddy
```