ArgWizard

ArgWizard is an argument parsing library for C which is designed to be incredibly easy to use, with a very non-steep learning curve for any `getopt' experienced developers.

This document goes over the entirety of every file and it's functionality definitions. This document took an extreme amount of time to write.

wError.h

This file defines simple functions which are used for error handling within ArgWizard.

Defines:

```
// a definition which frees any objects
// that are passed, as well as displays a
// fancy error message
#define wErrorDisplay(msg, ...)

// a definitition which is basically the same as
// the first one, i was just too
// lazy to interchange the comments
#define wErrorFreeAndDisplay(msg, obj)
```

Purpose:

This file defines 2 macros, one for displaying errors, and the other for displaying errors then freeing an object passed into it. The first one is primarily the main error used, as the second one has no real purpose and will most likely be deprecated in future versions.

wName.c

This file is the implementation for *wName.h.*

Contains definitions for telling apart types of arguments such as *–flag* and *-f.*

Defines:

```
/* defined in wName.h */
char *
wNameCreateFrom (char *name)

/* defined in wName.h */
wNameType
wNameTypeFromName (char *name)
```

Purpose:

To provide an easy way to analyze GNU-style command line arguments.

wName.h

Contains definitions for parsing command line flags.

Defines:

```
typedef enum _wNameType {
  WStraggly, // example: a.out, file.name
  WFlagSingular, // example: -f, -a
  WFlagLong, // example: --flag, --output
} wNameType;

// returns the name without
// the flag specifiers
char* wNameCreateFrom(char* name)

// Returns the type of the name
wNameType wNameTypeFromName (char *name)
```

Purpose:

To provide an easy way to analyze GNU-style command line arguments. (From *wName.c*) . . . – Continued on page 2

(Continued from page 1)

This header also defines an ENUM primarily focused on distinguishing the different kinds of flags that there are for command line arguments, such as -- & . Do note that *wName* is not able to distinguish key=value pairs, as that functionality is primarily handled by *wArgParser.c & wArgParser.h.*

wMem.h

Contains definitions for ArgWizard's memory pool, a simple list which keeps track of every memory pointer, in order to restore memory back to the OS once it is no longer in use.

NOTE: there are multiple memory pools which are passed around by member functions, all of which are freed in sequence. DO NOT attempt to free every memory pool by hand, instead, use *wArgParserDestroy()* to destroy every memory pool created by different child functions.

Defines:

```
struct wMemPool;
typedef struct wMemPool wMemPool;

// initializes the memory pool
wMemPool *wMemPoolCreate ();

// allocates memory on the memory
// pool and returns it, this is a
// simple way to keep track of all
// allocated memory and reduces the
// risk of memory leaks or fragmentation
void *wMemAlloc (wMemPool *self, size_t size);
```

(continued on page 2)

```
void *wMemRealloc(wMemPool *self,
                  void* ptr,
                  size_t size);
void wMemPoolDestroy (wMemPool *self);
```

Purpose:

ArgWizard contains a simple memory pooling system in order to keep track of every pointer of memory which is created. While, this is flexible, it leaves the problem of having one global memory pool, this is the reason why multiple objects may have an internal memory pool, which greatly reduces the amount of free calls that will have to be called in order to free the memory associated with them.

wMem.c

This file implements memory pooling, using `malloc', creating an expandable list of *void** memory pointers that can be created using *wMemAlloc()*

Defines:

```
struct wMemPool
{
    ...
};
// creates a memory pool,
// with a fixed-incremental
// size of members, that holds
// memory pointers.
wMemPool *
wMemPoolCreate ()

// allocates memory and returns it,
// also storing the pointer into the
// memory pool.
void *
wMemAlloc (wMemPool *self, size_t size)
```

(continued on page 2)

```
// this function creates a
// reallocated pointer and
// returns it, storing the
// reallocated pointer into
// the memory pool.
void *
wMemRealloc (wMemPool *self,
                void *ptr,
                size_t size)
// destroys a memory pool, freeing every
// pointer allocated to it.
void
wMemPoolDestroy (wMemPool *self)
```

Purpose:

This file defines a list of memory pointers which is good for keeping track of intense allocations, in which, freeing every instance becomes problematic and time-consuming. This prevents memory leaking while also being expandable using *WARGLIB_DEFAULT_INCREMENT.*

wStraggly.c

This file implements **stragglies**, or arguments which do not have a dash.

Defines:

```
struct wStraggly
{
  ...
};
// creates a straggly list on the
// memory pool OPTIONS
wStraggly *
wStragglyCreate (wMemPool *options)

// returns the size of a straggly list
int
wStragglySize (wStraggly *self)

// returns the straggly list
char **
wStragglies (wStraggly *self)
```

(continued on page 2)

```
// returns the straggly at the
// specified index
char *
wStragglyAt (wStraggly *self, int index)

// (deprecated, kept for compatibility)
// basically just wStragglyAt()
char *
wStragglyName (wStraggly *self, int index)

// appends a straggly to the list
void
wStragglyAppend (wMemPool *p,
                 wStraggly *self,
                 char *str)
```

Purpose:

Stragglies are arguments without a dash, or, just loose-leaf arguments, in any other argument parsing library. So something like specifying *`main.c'* in *gcc* is considered a straggly.