Documentation V1

June 2015

Contents

List of Figures

Chapter 1

library.c

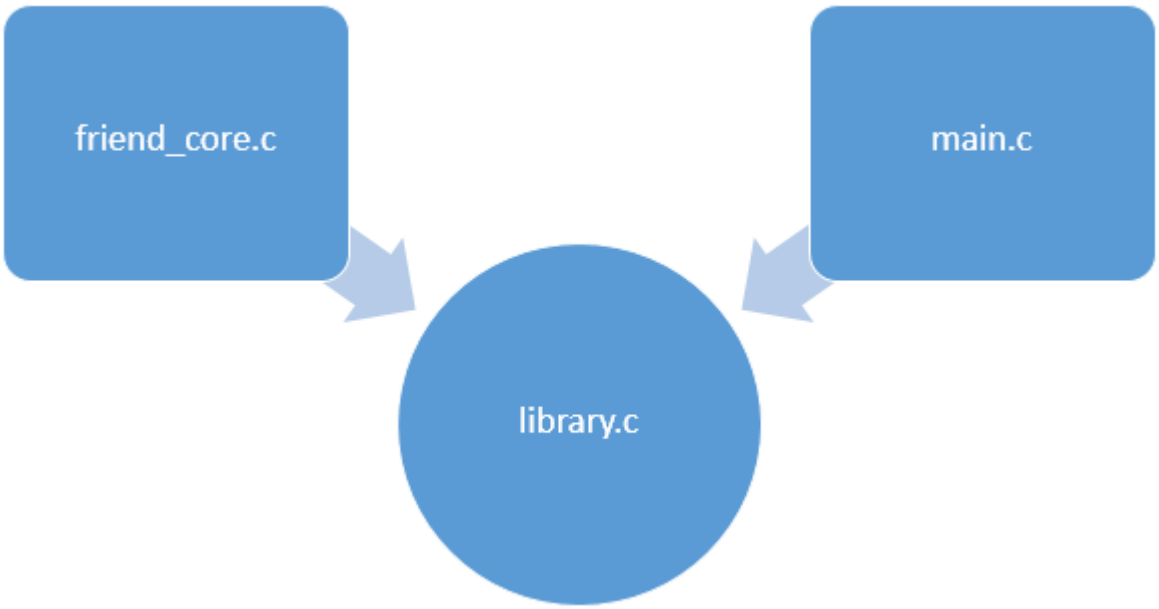

Figure 1.1: Calls to functions of library.c

1.1 LibraryOpen()

The LibraryOpen() function is used to open and initialize a Library struct, which may be used to call methods of the library opened. All libraries use the Library base struct, but may be extended depending on the library in use. The extended fields can always be found in the header file accompanying the library.

Declaration

struct Library* LibraryOpen(char* name, long version)

Parameters

char * name

A null-terminated string representing the name of the library to be loaded.

long version

An integer representing the lowest version of the library the callee is compatible with.

Returns

A pointer to a Library struct if the library was found. If the library cannot be found or an error occurred, a NULL pointer will be returned.

Example

```
// Open the library
Library_t* lib = LibraryOpen( "example.library", 1 );
if( lib == NULL )
{
    printf( "Couldn't load the library.\n" );
    return 1;
}
// Call a function of the library
long ver = lib->GetVersion( lib );
```

```
// Close the library
lib->libClose( lib );
```

See Also

- LibraryClose()

1.2 LibraryClose()

The LibraryClose() function is used to close the struct Library passed by the pointer and does not return any value.

Declaration

void LibraryClose(struct Library* library);

Parameters

struct Library *library

A struct pointer to the Library to be closed.

Returns

This function does not return any value.

Examples

```
// Open the library
Library_t* lib = LibraryOpen( "example.library", 1 );
if( lib == NULL )
{
    printf( "Couldn't load the library.\n" );
    return 1;
}
// Call a function of the library
long ver = lib->GetVersion( lib );
```

```
// Close the library
lib->libClose( lib );
```

See Also

- LibraryOpen()

Chapter 2

string.c

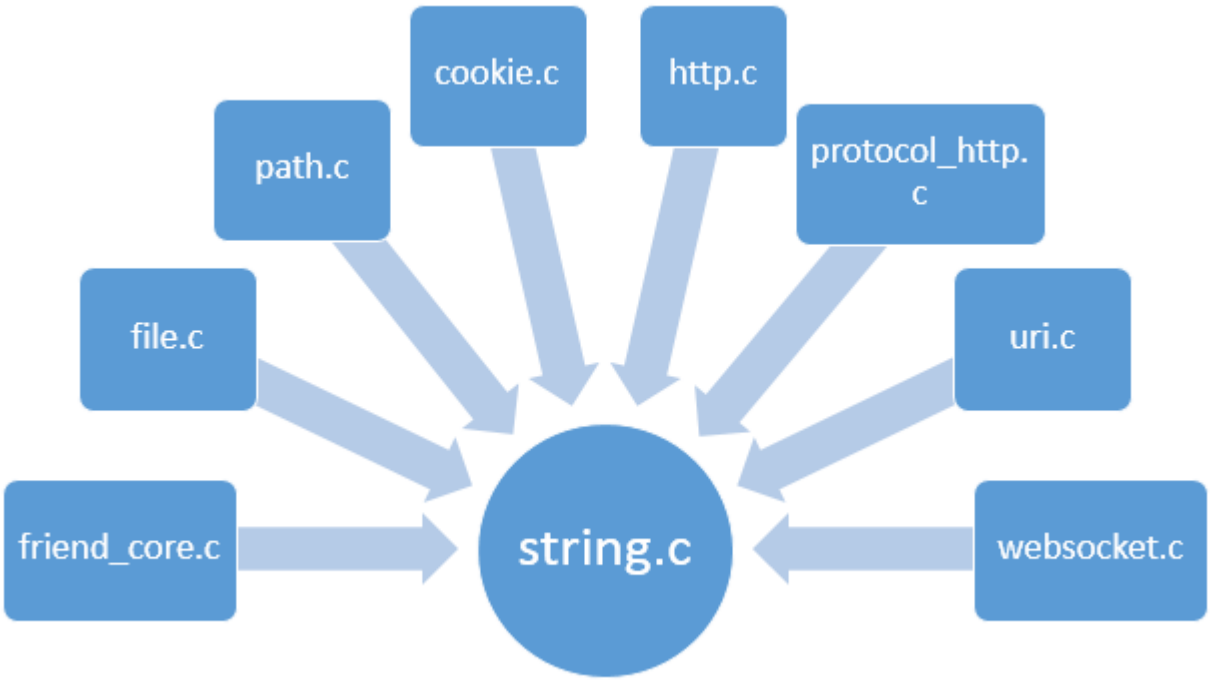

Figure 2.1: Calls to functions of string.c

2.1 MakeString()

The MakeString(int length) function allocates the requested memory passed by integer **length** and returns a pointer to the allocated memory. Each will assigned with a value of zero. It is equal to //return calloc (length + 1, sizeof (char));

Declaration

char* MakeString(int length)

Parameters

int length

This is the size of the memory should be allocated.

Returns

This function returns a pointer to the allocated memory, or returns NULL if the request fails.

Examples

```
char *str;
int len = strlen( str );
char *tmp = MakeString( len );
```

See Also

- SubStrCmp()
- SafeString()

2.2 SubStrCmp()

The function SubStrCmp(char *str, char *compare) finds first occurrence of string **compare** in string **str**.

Declaration

int SubStrCmp (char *str, char *compare)

Parameters

char *str

Pointer to the string where the first occurrence of the string pointed by pointer **compare**.

char *compare

Pointer to the string which should be find in the pointer string **str**.

Returns

The function returns the position of the first occurrence or it returns -1 if they do not match or either one of the input parameter is NULL.

Examples

```
char* var1="This is Friend UP";
char* var2="Friend";
//Compare the inputs to find the first occurance
int firstOccurance = SubStrCmp(var1,var2);
if(firstOccurance!=-1)
 {
    printf("First occurrence to string match at ");
    printf("%d", i);
 }else
 {
 printf("Input didn't match");
 }
```

```
// Above will write the number 8 to console.
```

See Also

- MakeString()

2.3 PStrlen()

The function PStrlen() gets the length of the passed string safely, even if the given string is NULL integer length 4 will be returned.

Declaration

int PStrlen (char *str)

Parameters

char *str

Pointer to the string which length should be measured.

Returns

Length of the passed string is returned, if the passed string is NULL, integer 4 is returned.

Examples

```
char* var1 ="Friend UP is Amazing";
char* var2=NULL;
int length = PStrlen(var1);
int lengthnull=PStrlen(var2);
printf("Length of the %s", var1);
printf(" is %d\n", length);
printf("Length of the %s", var2);
printf(" is %d",lengthnull);
getch();
```

```
return ;

//Above code print the following to console
Length of the Friend UP is Amazing is 20
Lenght of the <null> is 4
```

See Also

- SafeString()

2.4 SafeString()

The function SafeStrlen() finds the length in a safe way, if the string is not NULL terminated, it will be NULL terminated and will return the length.

Declaration

int SafeString (char **string, int length)

Parameters

char *string

Pointer to the string is given

int maxlen

The maximum length of the string is passed

Return

The length of the string if string properly null terminated, else fist string will be properly null terminated and length will be returned.

Examples

```
char* var1= "hello, world";
// Find the size of the string declared
int sizeOfString = sizeof (var1);
printf("Size of the String: %d\n", sizeOfString); // This will
    print 32 to console
// Find the length of the string
int lengthOfString = SafeString ( var1 );
printf("Lenght of the String: %d\n", lengthOfString); // This will
    print 12 to console
int lenghtOfNullString = PStrnln("");
printf("Lenght of the Null String: %d\n", lenghtOfNullString); //
    This will print 4 to console
```

See Also

- PStrlen()

2.5 StringDuplicateN()

The function StringDuplicateN() duplicates the string passed **str** with the length of passed length **len** and return the pointer to the duplicated string. If the length passed was 0, it returns NULL.

Declaration

char* StringDuplicateN(char* str, unsigned int len)

Parameters

char* str

This is pointer the string to be duplicated.

unsigned int len

Unsigned integer length specify the length of the string to be duplicated.

Return

This function returns a pointer to the duplicated string or NULL if the request fails or NULL if the passed length is 0.

Examples

```
char* str = "hello, world";

char* copied = StringDuplicateN(str, 10); // This will copy the
    First 10 characters and return the pointer to the creater string
printf("Copied String: %s\n", copied); // This will print the
    first 10 characters,
char* copiedTwo = StringDuplicateN( str, 0 ); // This will return
    NULL.
printf("Copied String: %s\n", copiedTwo); // This will print NULL;
getch();
return;

// above will print following
Copied String: hello, wor
Copied String: <null>
```

See Also

- StringDuplicate()

2.6 StringDuplicate()

The function StringDuplicate(const char* str) duplicates the string pointer by **str** and return the pointer to the duplicated string.

Declaration

char* StringDuplicate(const char* str)

Parameters

char* str

This is the pointer the string to be duplicated.

Return

The function returns the pointer to the duplicated string or NULL if fails.

Examples

```
char* str = "hello, world";

char* copied = StringDuplicate(str); // This will copy the First
    10 characters and return the pointer to the creater string
printf("Copied String: %s\n", copied); // This will print the
    first 10 characters,
getch();
return;
//Above code will print follwing,
Copied String: hello, world
```

See Also

- StringDuplicateN()

2.7 StringSecureFree()

The function void StringSecureFree(char* str) first overwrites the string with 0 which given by the pointer **str** and deallocates the memory previously allocated by a call to MakeString(). It is useful to re allocate the memory used to save the passwords.

Declaration

void StringSecureFree(char* str)

Parameters

This is the pointer to a memory block previously allocated with MakeString() to be deallocated. If a null pointer is passed as argument, no action occurs.

Return

This function does not return any value.

Examples

```
char *str;
/* Initial memory allocation */
str = (char *) malloc(8);
strcpy(str, "friendUP");
printf("String = %s, Address = %u\n", str, str);
/* Deallocate allocated memory */
StringSecureFree(str);
```

See Also

- StringDuplicateN()

2.8 StrLenSafeSpaces()

The function StrLenSafeSpaces(char *str) finds the length of the string, however ” ” became ” ”, and counted.

Declaration

int StrLenSafeSpaces(char *str)

Parameters

char *str

The pointer to the string which length should be found.

Returns

This function returns the length of the string without the space.

Examples

```
char* str="We are FriendUP";
int len;
len = strlen(str);
printf("Length of |%s| is |%d|\n", str, len);
len = StrLenSafeSpaces(str);
printf("Length after the spaces concerted to / of |%s| is |%d|\n",
    str, len);
getch();
return;
//You would expect output like follows
//Length of |We are FriendUP| is |15|
//Length after the spaces concerted to "\ " of |We are FriendUP|
    is |17|
```

See also

- StringDuplicateN()

2.9 CleanPathString()

The function CleanPathString(char *str) cleans the double slashes (//) from the string pointed by **str**.

Declaration

void CleanPathString(char *str)

Parameters

char *str

The pointer to the string the double slashes to be removed from.

Returns

This function does not return any value.

Examples

```
char str[50];
int len;
CleanPathString(str, "We are // FriendUP");
printf("The String after cleaned double slashes |%s| is |%d|\n",
    str);
//You would expect output like follows
//The String after cleaned double slashes |We are FriendUP|
```

See also

- AddEscapeChars()

2.10 AddEscapeChars()

The function AddEscapeChars(char *str) adds escape characters to the string pointed by **str** by replacing with
.

Declaration

void AddEscapeChars(char *str)

Parameters

char *str

This pointer point to the string Escape Character to be added

Returns

This function does not return any value.

Examples

```
char str[50];
int len;
AddEscapeChars(str, "We are FriendUP");
printf("The String after Escape character added is: %s\n", str);
//You would expect output like follows
//The String after Escape character added is: We\\are\\FriendUP
```

See Also

- CleanPathString()

2.11 StringParseUInt()

The Method StringParseUInt(char* str) parses the number passed as ASCII string pointer by **str** and return the unsigned integer, If there is a non-digit character in the passed string, 0 is returned. If the number is too large to fit in an uint, this function will simply overflow.

Declaration

unsigned int StringParseUInt(char* str)

Parameters

char* str

The pointer to the ASCII denoted number

Returns

Unsigned integer represented by the character passed is returned, if the character passed was non-digit character 0 is returned. if the number is too long, function will over flow.

Examples

```
char str[32] = "55";
char strNonDigit[32]="S"
StringParseUInt(str); // Output will be 55,
StringParseUInt(strNonDigit); // Output will be 0,
```

See Also

- CleanPathString()

2.12 StringAppend()

The function StringAppend(const char *src, const char *add) appends the string pointed by **add** to string pointed by **src**.

Declaration

char* StringAppend(const char *src, const char *add)

Parameters

const char *src

The pointer to the string that need to be appended with.

const char *add

The pointer to the string need to be appended

Results

Pointer to the Appended string is returned

Example

```
char str[32] = "Friend UP";
char strAppend[32]="is Amazing"
StringAppend(str,strAppend); // Out put will be "Friend UP is
    Amazing,
```

See Also

- CleanPathString()

2.13 CharIsDigit()

The function CharIsDigit(char c) checks whether passed char is a digit or not, bool value true is returned when the character passed is digit, else it returns false.

Declaration

bool CharIsDigit(char c)

Parameters

char c

The character to be checked whether digit or not.

Return

This function returns true if the char passed is a digit, else it returns false.

Example

```
char chartocheck1='5';
char chartocheck='A';

if(CharIsDigit(chartocheck))
```

```
{
printf("%c", chartocheck);
printf(" is a Digit\n");
}else
{
printf("%c", chartocheck);
printf(" is not a Digit\n");
}
if(CharIsDigit(chartocheck1))
{
printf("%c", chartocheck1);
printf(" is a Digit\n");
}else
{
printf("%c", chartocheck1);
printf(" is not a Digit\n");
}
// This will print following to console
// A is not a Digit
// 5 is a Digit
```

See Also

- CharIsUpAlpha()
- CharIsLoAlpha()
- CharIsAlpha()
- CharIsAlphanumeric()

2.14 CharIsUpAlpha()

The Method CharIsUpAlpha() checks whether passed char is upper case Alpha (i.e A,B,.. etc.) or not, bool value true is returned when the character passed is upper case Alpha, else returns false.

Declaration

bool CharIsUpAlpha(char c)

Parameters

char c

The character to be checked whether upper case Alpha or not.

Return

This function returns true if the char passed is uppercase Alpha, else returns false.

Example

```
char chartocheck1='5';
char chartocheck='A';

if(CharIsUpAlpha(chartocheck))
{
   printf("%c", chartocheck);
   printf(" is a uppercase alphabet\n");
}else
{
   printf("%c", chartocheck);
   printf(" is not a uppercase alphabet\n");
}
if(CharIsUpAlpha(chartocheck1))
{
   printf("%c", chartocheck1);
   printf(" is a uppercase alphabet\n");
}else
{
   printf("%c", chartocheck1);
   printf(" is not a uppercase alphabet\n");
}
// This will print following to console
// A is a uppercase alphabet
```

```
// 5 is not a uppercase alphabet
```

See Also

- CharIsDigit()
- CharIsLoAlpha()
- CharIsAlpha()
- CharIsAlphanumeric()

2.15 CharIsLoAlpha()

The function CharIsLoAlpha(char c) checks whether passed char is lower case alpha(a,b,.. etc.) or not, bool value true is returned when the character passed is lower case alpha, else it returns false.

Declaration

bool CharIsLoAlpha(char c)

Parameters

char c

The character to be checked whether lower case alpha or not

Return

This function returns true if the char passed is lower case alpha, else it returns false.

Example

```
char chartocheck='A';
char chartocheck1='a';

if(CharIsUpAlpha(chartocheck))
{
   printf("%c", chartocheck);
   printf(" is a lowercase alphabet\n");
}else
{
   printf("%c", chartocheck);
   printf(" is not a lowercase alphabet\n");
}
if(CharIsUpAlpha(chartocheck1))
{
   printf("%c", chartocheck1);
   printf(" is a lowercase alphabet\n");
}else
{
   printf("%c", chartocheck1);
   printf(" is not a lowercase alphabet\n");
}
// This will print following to console
// A is not a lowercase alphabet
// a is a lowercase alphabet
```

See Also

- CharIsDigit()
- CharIsUpAlpha()
- CharIsAlpha()
- CharIsAlphanumeric()

2.16 CharIsAlpha()

The function CharIsAlpha(char c) checks whether passed char is Alpha(A,B,C,a,b.. etc.) or not, bool value true is returned when the character passed is alpha, else it returns false.

Declaration

bool CharIsAlpha(char c)

Parameters

char c

The character to be checked whether Alpha or not.

Returns

This function returns true if the char passed is Alpha, else it returns false.

Example

```
char chartocheck='A';
char chartocheck1='a';
char chartocheck2='5';

if(CharIsAlpha(chartocheck))
{
   printf("%c", chartocheck);
   printf(" is an alphabet\n");
}else
{
   printf("%c", chartocheck);
   printf(" is not an alphabet\n");
}
if(CharIsAlpha(chartocheck1))
{
   printf("%c", chartocheck1);
   printf(" is an alphabet\n");
```

```
}else
{
   printf("%c", chartocheck1);
   printf(" is not an alphabet\n");
}
if(CharIsAlpha(chartocheck1))
{
   printf("%c", chartocheck1);
   printf(" is an alphabet\n");
}else
{
   printf("%c", chartocheck1);
   printf(" is not an alphabet\n");
}
// This will print following to console
// A is an alphabet
// a is an alphabet
// 5 is not an alphabet
```

See Also

- CharIsDigit()
- CharIsUpAlpha()
- CharIsDigit()
- CharIsAlphanumeric()

2.17 CharIsAlphanumeric()

The function CharIsAlphanumeric(char c) checks whether passed char is Alpha(A,B,C,c.. etc.) or Numeric (1,3,..etc.), bool value true is returned when the character passed is Alpha or numeric, else it returns false.

Declaration

bool CharIsAlphanumeric(char c)

Parameters

char c

The character to be checked whether l Alpha or Numeric

Return

This method return true if the char passed is Alpha or Numeric, else it returns false.

Example

```
char chartocheck='A';
char chartocheck1='a';
char chartocheck2='5';

if(CharIsAlphanumeric(chartocheck))
{
printf("%c", chartocheck);
printf(" is an alphabet or numeric\n");
}else
{
printf("%c", chartocheck);
printf(" is not an alphabet or numeric\n");
}
if(CharIsAlphanumeric(chartocheck1))
{
printf("%c", chartocheck1);
printf(" is an alphabet or numeric\n");
}else
{
printf("%c", chartocheck1);
printf(" is not an alphabet or numeric\n");
}
if(CharIsAlphanumeric(chartocheck1))
{
printf("%c", chartocheck1);
printf(" is an alphabet or numeric\n");
}else
```

```
{
printf("%c", chartocheck1);
printf(" is not an alphabet or numeric\n");
}
// This will print following to console
// A is an alphabet or numeric
// a is an alphabet or numeric
// 5 is an alphabet or numeric
```

See Also

- CharIsDigit()
- CharIsUpAlpha()
- CharIsAlpha()
- CharIsAlphanumeric()

2.18 CharAlphaToUp()

The function CharAlphaToUp(char c) converts the passed lowercase character to uppercase character and converted character returned.

Declaration

char CharAlphaToUp(char c)

Parameters

char c

The character to be changed to upper case

Return

It returns uppercase character.

Example

```
char chartocheck='a';
char changed = CharAlphaToUp(chartocheck);

printf("%c", chartocheck);
printf(" is changed as ");
printf("%c", changed);

//This will print as follows
// a is changed as A
```

See Also

- StringToUppercase()
- StringToLowercase()
- CharAlphaToLow()

2.19 CharAlphaToLow()

The function CharAlphaToLow(char c) converts the passed uppercase character to lowercase character and converted character returned.

Declaration

char CharAlphaToLow(char c)

Parameters

char c

The character to be changed to lowercase.

Returns

The changed lower case character is returned

Example

```
 char chartocheck='A';
char changed = CharAlphaToUp(chartocheck);

 printf("%c", chartocheck);
printf(" is changed as ");
printf("%c", changed);

//This will print as follows
// A is changed as a
```

See Also

- StringToUppercase()
- StringToLowercase()
- CharAlphaToLow()

2.20 CharIsCTL()

The method CharIsCTL(char c) checks whether passed character is control character.

Declaration

bool CharIsCTL(char c)

Parameters

char c

The character to be checked whether control character or not

Return

This method is returned true if passed character is control character

Example

```
char var1 = 'Ctrl-@';
char var2 = 'Ctrl-C';
char var3 = 'M';

if( CharIsCTL(var1) )
{
printf("var1 = |%c| is control character \n", var1 );
}
else
{
printf("var1 = |%c| is not control character\n", var1 );
}

if( CharIsCTL(var2) )
{
printf("var2 = |%c| is control character\n", var2 );
}
else
{
printf("var2 = |%c| is not control character\n", var2 );
}

if( CharIsCTL(var3) )
{
printf("var3 = |%c| is control character\n", var3 );
}
else
{
printf("var3 = |%c| is not control character\n", var3 );
}

//Output will be look like this

var1 = |Ctrl-@| is control character
var2 = |Ctrl-c| is control character
var3 = |M| is not control character
```

See Also

- StringToLowercase()
- CharAlphaToUp()
- CharAlphaToLow()

2.21 StringToLowercase()

The function StringToLowercase(char* str) is used to convert the string pointer by the pointer **str** to lowercase string.

Declaration

void StringToLowercase(char* str)

Parameters

char* str

The pointer to the string to be changed to lower case

Return

This function does not return anything.

Example

```
char* str="Friend UP";
char* changed = StringToLowercase(str);

 printf("%c", str);
printf(" is changed as ");
printf("%c", changed);

//This will print as follows
// Friend UP is changed as friend up
```

See also

- StringToUppercase()
- CharAlphaToUp()
- CharAlphaToLow()

2.22 StringToUppercase()

The function StringToUppercase(char* str) is used to convert string pointer by the pointer **str** to uppercase string.

Declaration

void StringToUppercase(char* str)

Parameters

char* str

The pointer to the string to be changed to upper case

Return

This method doesnt return anything

Example

```
char* str="Friend UP"
StringToUppercase(str); // Return FRIEND UP
```

See also

- StringToLowercase()
- CharAlphaToUp()
- CharAlphaToLow()

2.23 StringCheckExtension()

The function StringCheckExtension(char* str, char* ext) compare the extension given by the pointer **ext** with string given by the pointer **str** and return the integer values based on following condition.

if str1 is less than str2 then return value ¡ 0;

if str2 is less than str1 then return value ¿ 0;

if str1 is equal to str2 then return value =0;

Declaration

int StringCheckExtension(char* str, char* ext)

Parameters

char* str

The string where extension need to be checked.

char* ext

The pointer to the extension string.

Return

This function return integer value based on the following rules,
if str1 is less than str2 then return value ¡ 0;

if str2 is less than str1 then return value ¿ 0;

if str1 is equal to str2 then return value =0;

Example

```
char* str="FriendUP.com";
   char* ext="com";
   int index=StringCheckExtension(str,ext);
   if(index==0)
   {
      printf("The extension supplied(ext) and the extension in the
          str are same");
      getch();
      return;
   }
   printf("The extension supplied(ext) and the extension in the str
       are not same");
   // this will print as follows
   The extension supplied(ext) and the extension in the str are same
```

See also

- StringToLowercase()
- CharAlphaToUp()
- CharAlphaToLow()

2.24 UrlDecode()

The function UrlDecode(char* dst, const char* src) decodes the string passed by pointer **src**.

Declaration

ptrdiff_t UrlDecode(char* dst, const char* src)

Parameters

char* dst

const char* src

Return

Example

See also

- CharAlphaToLow()

2.25 StringSplit()

The function StringSplit(char* str, char delimiter, unsigned int* length) spit the string into list of strings using the delimiter **demimiter**. A null pointer is returned if there are no delimiter **demimiter** found in the passed string **str**.

Declaration

char** StringSplit(char* str, char delimiter, unsigned int* length)

Parameters

char* str

The contents of this string are broken into smaller strings using the delimiter.

char delimiter

his is the C string containing the delimiters.

unsigned int* length

Return

This function returns a pointer to the last token found in the string. A null pointer is returned if there are no tokens left to retrieve.

Example

See also

- CharAlphaToLow()

2.26 StringShellEscape()

The function StringShellEscape(const char* str) removes the and ” from the passed string **str** and return the pointer to the character and ” removed string.

Declaration

char* StringShellEscape(const char* str)

Parameters

const char* str

This is pointer to the string where the characters to be removed.

Return

Returns the pointer to the string where the characters removed.

Example

See also

- CharAlphaToLow()

Chapter 3

http.c

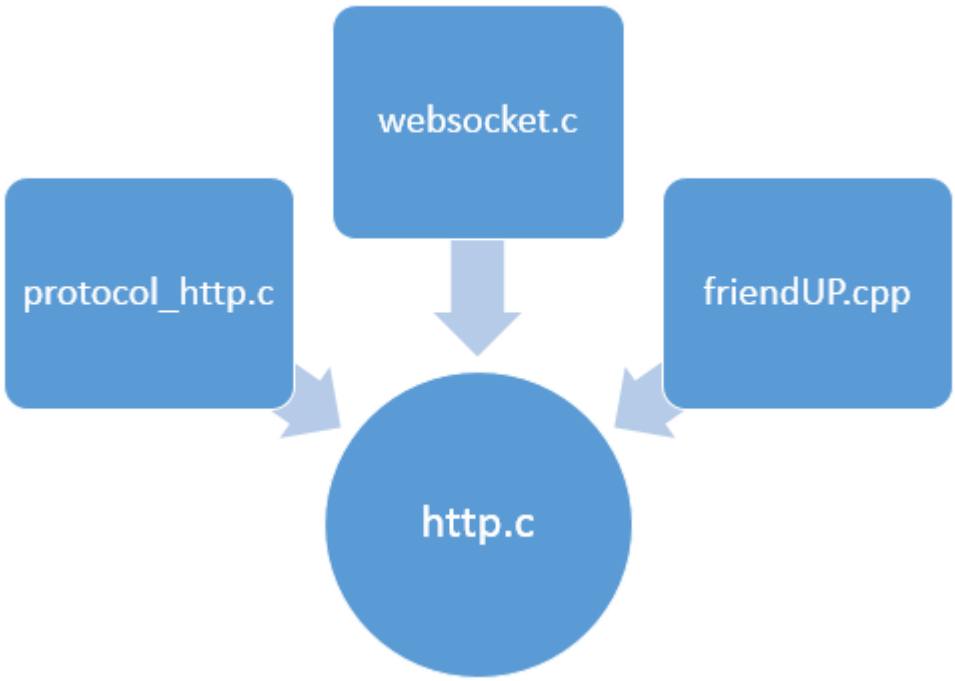

Figure 3.1: Calls to functions of http.c

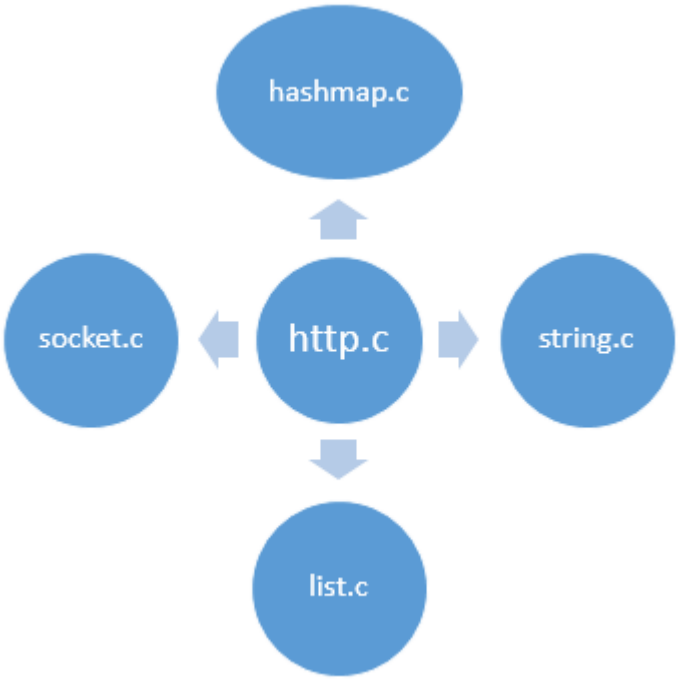

Figure 3.2: Calls from http.c

3.1 HttpNew()

The function HttpNew() allocates the memory in the size of struct Http_t, and returns the pointer to the created Http_t object. The fields of struct Http_t are set to their default values - NULL, and the fields versionMajor and versionMinor are set to 1. This function is used in HttpNewSimple() in order to allocate memory to the correct size of struct Http_t. This function is used to create the Http response or the request in the Http or web socket class.

Declaration

Http_t* HttpNew()

Parameters

This function does not have any parameters.

Returns

This function returns a pointer to the created Htt_t object.

Examples

```
int main ()
{
   struct Http *http;
   http = HttpNew();
   return(0);
}
```

```
void ProtocolHttp( Socket_t* sock, char* data, unsigned int length
    )
{
   // Get the current request we're working on, or start a new one
   Http_t* request = (Http_t*)sock->data;
   if( !request )
   {
      request = HttpNew();
      request->timestamp = time( NULL );
      sock->data = (void*)request;
   }
}
\caption{Function used to create the request from the socket data}
```

See Also

- HttpNewSimple()

3.2 HttpNewSimple()

The function HttpNewSimple(unsigned int code, unsigned int numHeaders, ...) allocates the memory in the size of struct Http_t, and returns the pointer to the created Http_t object. The fields of struct Http_t are set to their default values - NULL, and the fields versionMajor and versionMinor are set to 1. The field **responseCode** is set to the value passed by the **code** and the respective reason is set to the **responseReason** field. This function can be used to create the custom and error http response such as *404 Not found*, etc.

Declaration

Http_t* HttpNewSimple(unsigned int code, unsigned int numHeaders, ...)

Parameters

unsigned int code

An integer representing the response code.

unsigned int numHeaders

An integer representing the number of http headers to be added with the default values in the header.

Returns

This function returns a pointer to created Http_t object.

Examples

```
void Http404( Socket_t* sock )
{
   Http_t* response = HttpNewSimple( HTTP_404_NOT_FOUND,
       4,"Content-Type", StringDuplicate( "text/plain" ),
       "Connection", StringDuplicate( "close" ));
   HttpAddTextContent( response, "404 Not Found\n" );
   HttpWriteAndFree( response, sock );
```

```
   SocketShutdown( sock );
}
```

See Also

- HttpNew()

3.3 HttpError()

The function HttpError(unsigned int code, Http_t* http, unsigned int line) adds the passed error code to the Http_t object, adds the error line passed by the integer value **line** and return the pointer to the Http_t object which was passed to the function.

Declaration

Http_t* HttpError(unsigned int code, Http_t* http, unsigned int line)

Parameters

unsigned int code

An unsigned integer representing the error.

Http_t* http

A Http_t pointer where the error code and error line should be set.

unsigned int line

An unsigned integer representing the line of error.

Returns

A pointer to the Http_t object, which error code and error line is updated.

See Also

- HttpNew()

3.4 HttpIsChar()

The function HttpIsChar() checks whether the passed character **c** is a US-ASCII character. The US-ASCII character are falls in the range of octets 0 - octets 127.

Declaration

bool HttpIsChar(char c)

Parameters

char c

A character which should be checked for US-ASCII.

Returns

This function returns a non-zero value(true) if passed character is a US-ASCII character, else returns zero (false).

Examples

```
char var1 = 'M';
char var2 = 'm';
char var3 = '3';

if( HttpIsChar(var1) )
{
   printf("var1 = |%c| is US-ASCII character \n", var1 );
}
else
{
   printf("var1 = |%c| is not US-ASCII character\n", var1 );
```

```
}

if( HttpIsChar(var2) )
{
   printf("var2 = |%c| is US-ASCII character\n", var2 );
}
else
{
   printf("var2 = |%c| is not US-ASCII character\n", var2 );
}

if( HttpIsChar(var3) )
{
   printf("var3 = |%c| is US-ASCII character\n", var3 );
}
else
{
   printf("var3 = |%c| is not US-ASCII character\n", var3 );
}

//Output will be look like this

var1 = |M| is US-ASCII character
var2 = |m| is US-ASCII character
var3 = |3| is US-ASCII character
```

See Also

- HttpIsCTL()

3.5 HttpIsCTL()

The function HttpIsCTL(char c) checks whether the passed character **c** is a control character. The US-ASCII control characters are (octets 0 - 31) and DEL (127)

Declaration

bool HttpIsCTL(char c)

Parameters

char c

A character which should be checked for control character.

Returns

This function returns a non-zero value(true) if c is a control character, else returns zero (false).

Examples

```
char var1 = 'Ctrl-@';
char var2 = 'Ctrl-C';
char var3 = 'M';

if( HttpIsCTL(var1) )
{
   printf("var1 = |%c| is US-ASCII control character \n", var1 );
}
else
{
   printf("var1 = |%c| is not US-ASCII control character\n",
       var1 );
}

if( HttpIsCTL(var2) )
{
   printf("var2 = |%c| is US-ASCII control character\n", var2 );
}
else
{
   printf("var2 = |%c| is not US-ASCII control character\n",
       var2 );
}
```

```
if( HttpIsCTL(var3) )
{
   printf("var3 = |%c| is US-ASCII control character\n", var3 );
}
else
{
   printf("var3 = |%c| is not US-ASCII control character\n",
       var3 );
}

//Output will be look like this

var1 = |Ctrl-@| is US-ASCII control character
var2 = |Ctrl-c| is US-ASCII control character
var3 = |M| is not US-ASCII control character
```

See Also

- HttpIsChar()
- HttpIsSeparator()
- HttpIsUpAlpha()

3.6 HttpIsSeparator()

The function HttpIsSeparator(char c) checks whether the passed character **c** is a separator character. The separator character set is as follows:

" " ( ) < > @
, ; " / [
] ? = { } 0*09

Declaration

bool HttpIsSeparator(char c)

Parameters

char c

A character which should be checked for separator character.

Returns

This function returns a non-zero value(true) if c is a separator character, else returns zero (false).

Examples

```
char var1 = '[';
char var2 = '}';
char var3 = ';';
char var4 = 'A';


if( HttpIsSeparator(var1) )
{
   printf("var1 = |%c| is separator character \n", var1 );
}
else
{
   printf("var1 = |%c| is not separator character\n", var1 );
}

if( HttpIsSeparator(var2) )
{
   printf("var2 = |%c| is separator character\n", var2 );
}
else
{
   printf("var2 = |%c| is not separator character\n", var2 );
}

if( HttpIsSeparator(var3) )
{
   printf("var3 = |%c| is separator character\n", var3 );
```

```
}
else
{
   printf("var3 = |%c| is not separator character\n", var3 );
}
if( HttpIsSeparator(var4) )
{
   printf("var4 = |%c| is separator character\n", var3 );
}
else
{
   printf("var4 = |%c| is not separator character\n", var3 );
}

//Output will be look like this

var1 = |[| is separator character
var2 = |}| is separator character
var3 = |;| is separator character
var4 = |A| is not separator character
```

See Also

- HttpIsChar()
- HttpIsLoAlpha()
- HttpIsUpAlpha()
- HttpIsCTL()

3.7 HttpIsUpAlpha()

The function HttpIsUpAlpha(char c) is checks whether the passed character **c** is a uppercase character.

Declaration

bool HttpIsUpAlpha(char c)

Parameters

char c

A character which should be checked for uppercase character.

Returns

This function returns a non-zero value(true) if c is a uppercase character, else returns zero (false).

Examples

```
char var1 = 'M';
char var2 = 'm';
char var3 = '3';

if( HttpIsUpAlpha(var1) )
{
   printf("var1 = |%c| is uppercase character\n", var1 );
}
else
{
   printf("var1 = |%c| is not uppercase character\n", var1 );
}

if( HttpIsUpAlpha(var2) )
{
   printf("var2 = |%c| is uppercase character\n", var2 );
}
else
{
   printf("var2 = |%c| is not uppercase character\n", var2 );
}

if( HttpIsUpAlpha(var3) )
{
   printf("var3 = |%c| is uppercase character\n", var3 );
}
else
```

```
{
   printf("var3 = |%c| is not uppercase character\n", var3 );
}

//The above code will print to console like as follows:
var1 = |M| is uppercase character
var2 = |m| is not uppercase character
var3 = |3| is not uppercase character
```

See Also

- HttpIsChar()
- HttpIsLoAlpha()
- HttpIsUpAlpha()
- HttpIsCTL()

3.8 HttpIsLoAlpha()

The function HttpIsLoAlpha(char c) checks whether the character passed is lowercase character.

Declaration

bool HttpIsLoAlpha(char c)

Parameters

char c

A character which should be checked for lowercase character.

Returns

This function returns a non-zero value(true) if c is a lowercase character, else returns zero (false).

Examples

```
char var1 = 'm';
char var2 = 'M';
char var3 = '3';

if( HttpIsLoAlpha(var1) )
{
   printf("var1 = |%c| is lowercase character\n", var1 );
}
else
{
   printf("var1 = |%c| is not lowercase character\n", var1 );
}

if( HttpIsLoAlpha(var2) )
{
   printf("var2 = |%c| is lowercase character\n", var2 );
}
else
{
   printf("var2 = |%c| is not lowercase character\n", var2 );
}

if( HttpIsLoAlpha(var3) )
{
   printf("var3 = |%c| is lowercase character\n", var3 );
}
else
{
   printf("var3 = |%c| is not lowercase character\n", var3 );
}

//The above code will print to console like as follows:
var1 = |m| is lowercase character
var2 = |M| is not lowercase character
var3 = |3| is not lowercase character
```

See Also

- HttpIsChar()
- HttpIsAlpha()
- HttpIsUpAlpha()
- HttpIsCTL()

3.9 HttpIsAlpha()

The function HttpIsAlpha(char c) checks whether the character passed is an alphabetic character.

Declaration

bool HttpIsAlpha(char c)

Parameters

char c

A character which should be checked for alphabetic character.

Returns

This function returns a non-zero value(true) if c is an alphabetic character, else returns zero (false).

Examples

```
char var1 = 'm';
char var2 = 'M';
char var3 = '3';

if( HttpIsAlpha(var1) )
{
   printf("var1 = |%c| is an alphabetic character\n", var1 );
}
```

```
else
{
   printf("var1 = |%c| is not an alphabetic character\n", var1 );
}

if( HttpIsAlpha(var2) )
{
   printf("var2 = |%c| is an alphabetic character\n", var2 );
}
else
{
   printf("var2 = |%c| is not an alphabetic character\n", var2 );
}

if( HttpIsAlpha(var3) )
{
   printf("var3 = |%c| is an alphabetic character\n", var3 );
}
else
{
   printf("var3 = |%c| is not an alphabetic character\n", var3 );
}

//The above code will print to console like as follows:
var1 = |m| is an alphabetic character
var2 = |M| is an alphabetic character
var3 = |3| is not an alphabetic character
```

See Also

- HttpIsChar()
- HttpIsLoAlpha()
- HttpIsUpAlpha()
- HttpIsCTL()

3.10 HttpAlphaToLow()

The function HttpAlphaToLow(char c) changes the passed uppercase alphabetic character to lowercase alphabetic character.

Declaration

char HttpAlphaToLow(char c)

Parameters

char c

A character to be changed to lowercase character.

Returns

A lowercase character returned.

Examples

```
char var1 = 'm';
char var2 = 'M';

char var3 = HttpAlphaToLow ( var1 );
char var4 = HttpAlphaToLow ( var 2);


printf("var1 = Before the change |%c|\n", var1 );
printf("var3 = After the change |%c|\n", var3 );
printf("var2 = Before the change |%c|\n", var2 );
printf("var4 = After the change |%c|\n", var4 );

// This will print the following to console,
var1 = |m| = Before the change
var3 = |m| = After the change
var2 = |M| = Before the change
var4 = |m| = After the change
```

See Also

- HttpIsChar()
- HttpIsLoAlpha()
- HttpIsUpAlpha()
- HttpIsCTL()

3.11 HttpIsToken()

The function HttpIsToken(char c) checks whether the character passed is a not a control character and separator character, but a character.

Declaration

bool HttpIsToken(char c)

Parameters

char c

A character which should be checked for Token character.

Returns

This function returns a non-zero value(true) if c is a control character and separator character, but a character, else returns zero (false).

Examples

```
char var1 = ';';
char var2 = 'Ctrl-@';
char var3 = 'M';

if( HttpIsToken(var1) )
{
   printf("var1 = |%c| is a token character\n", var1 );
}
```

```
else
{
   printf("var1 = |%c| is not a token character\n", var1 );
}

if( HttpIsToken(var2) )
{
   printf("var2 = |%c| is a token character\n", var2 );
}
else
{
   printf("var2 = |%c| is not a token character\n", var2 );
}

if( HttpIsToken(var3) )
{
   printf("var3 = |%c| is a token character\n", var3 );
}
else
{
   printf("var3 = |%c| is not a token character\n", var3 );
}

//The above code will print to console like as follows:
var1 = |;| is not a token character
var2 = |Ctrl-@| is not a token character
var3 = |M| is a token character
```

See Also

- HttpIsChar()
- HttpIsLoAlpha()
- HttpIsUpAlpha()
- HttpIsCTL()

3.12 HttpIsWhitespace()

The function HttpIsWhitespace(char c) checks whether the character passed is a whitespace.

Declaration

bool HttpIsWhitespace(char c)

Parameters

char* c

A character which should be checked for white space character.

Returns

This function returns a non-zero value(true) if c is a space character, else returns zero (false).

Examples

```
char var1 = ' ';
char var2 = '\t';
char var3 = 'M';

if( HttpIsToken(var1) )
{
   printf("var1 = |%c| is a space character\n", var1 );
}
else
{
   printf("var1 = |%c| is not a space character\n", var1 );
}

if( HttpIsToken(var2) )
{
   printf("var2 = |%c| is a space character\n", var2 );
}
```

```
else
{
   printf("var2 = |%c| is not a space character\n", var2 );
}

if( HttpIsToken(var3) )
{
   printf("var3 = |%c| is a space character\n", var3 );
}
else
{
   printf("var3 = |%c| is not a space character\n", var3 );
}

//The above code will print to console like as follows:
var1 = | | is a space character
var2 = |\t| is a space character
var3 = |3| is not a space character
```

See Also

- HttpIsChar()
- HttpIsLoAlpha()
- HttpIsUpAlpha()
- HttpIsCTL()
- HttpIsToken()

3.13 HttpParseInt()

The function HttpParseInt(char* str) parse the string passed by the pointer **str** into integer.

Declaration

int HttpParseInt(char* str)

Parameters

char* str

A pointer to the string to be parsed as integer.

Returns

The function returns the signed integer value of the string passed.

See Also

- HttpParseHeader()

3.14 HttpParseHeader()

The HttpParseHeader(Http_t* http, const char* request, unsigned int length) parse the http header passed as string by the pointer **request** and set the fields on the Http_t object passed by the pointer **http**. It return integer **400** if parsing failed. It returns 1 if the parsing successful. This function is used in the **HttpParsePartialRequest()** function.

Declaration

int HttpParseHeader(Http_t* http, const char* request, unsigned int length)

Parameters

Http_t* http

A pointer to the Http_t object where the fields should be set using the the header passed as string.

const char* request

A pointer to the string which representing the http header.

unsigned int length

A unsigned integer representing the length of the header string passed.

Returns

A signed integer returned based on the success of the request.

See Also

- HttpParsePartialRequest()

3.15 HttpParsePartialRequest()

The function HttpParsePartialRequest(Http_t* http, char* data, unsigned int length) parses the http partial request.

Declaration

int HttpParsePartialRequest(Http_t* http, char* data, unsigned int length)

Parameters

Http_t* http

A pointer to the Http_t object where the fields should be set using the the header passed as string.

const char* request

A pointer to the string which representing the http header.

unsigned int length

A unsigned integer representing the length of the header string passed.

Returns

A signed integer returned based on the success of the request.

See Also

- HttpParseHeader()

3.16 HttpGetHeaderList()

The function HttpGetHeaderList(Http_t* http, const char* name) get the raw list of data the pointed by the key passed by pointer **name** from the header list of the Http_t object passed by the pointer **http**.

Declaration

List_t* HttpGetHeaderList(Http_t* http, const char* name)

Parameters

Http_t* http

This is the Http_t object where list of http header has to be retrieved.

const char* name

This is the key associated with the list of header to be retrieved.

Returns

The function returns the list of http raw headers if found any, else NULL.

See Also

- HttpHeaderContains()

3.17 HttpNumHeader()

The function HttpNumHeader(Http_t* http, const char* name) returns the number of elements **http** contains which is associated with the key passed by **name**, returns 0 if there are no values.

Declaration

unsigned int HttpNumHeader(Http_t* http, const char* name)

Parameters

Http_t* http

A pointer to the Http_t object, where the count of values to associated with the key **name**.

const char* name

Returns

This function returns the number of values header contains, returns 0 if there are no values.

See Also

- HttpHeaderContains()

3.18 HttpHeaderContains()

The function HttpHeaderContains(Http_t* http, const char* name, const char* value, bool caseSensitive)checks whether the Http_t object **http** contains the value passed by **value** in the header list associated with the key **name**.

Declaration

bool HttpHeaderContains(Http_t* http, const char* name, const char* value, bool caseSensitive)

Parameters

Http_t* http

A pointer to the Http_t object where availability of the value to be checked.

const char* name

A string pointer referring the key of the http header list.

const char* value

A pointer referring the value should be checked in the http header list.

bool caseSensitive

A boolean value true or false denotes whether value should be checked with case sensitive.

Returns

This function returns true if value found in the header, else returns false.

See Also

- HttpNumHeader()

3.19 HttpFree()

The function HttpFree(Http_t* http) first deallocates the memory previously allocated to store the data and key of the http header list, and deallocates the memory previously allocated to store the http headers, and if it is a http response then it deallocated the memory used to store the response and set the response to NULL, and deallocates the memory used to store the content and set the content to null and finally deallocates the memory previously allocated to store the struct **Http_t**. This function can be used to free the Http object headers once the content is written to the socket.

Declaration

void HttpFree(Http_t* http)

Parameters

Http_t* http

A pointer to the Http_t object which memory should be deallocated.

Returns

This functions doesn't return any values.

Examples

```
void HttpWriteAndFree( Http_t* http, Socket_t* sock )
{
   HttpBuild( http );
   SocketWrite( sock, http->response, http->responseLength );
   HttpFree( http );
}
```

See Also

- HttpFreeRequest()

3.20 HttpFreeRequest()

The function HttpFreeRequest(Http_t* http) first deallocates the memory previously allocated to store the raw data such as *method*, *version*, etc.then deallocates the memory previously allocated to store the data and key of the http header list, and deallocates the memory previously allocated to store the http headers, and if it is a http response then it deallocated the memory used to store the response and set the response to NULL, and deallocates the

memory used to store the content and set the content to null and finally deallocates the memory previously allocated to store the struct. This function can be used once the socket used to write the response has been shutdown.

Declaration

void HttpFreeRequest(Http_t* http)

Parameters

Http_t* http

A pointer to the Http_t object which memory should be deallocated.

Returns

This function doesn't return any values.

Examples

```
if( time( NULL ) > request->timestamp + HTTP_REQUEST_TIMEOUT )
   {
      Http_t* response = HttpNewSimple(
         HTTP_408_REQUEST_TIME_OUT, 4,
         "Content-Type", StringDuplicate( "text/plain" ),
         "Connection", StringDuplicate( "close" )
      );
      HttpAddTextContent( response, "408 Request Timeout\n" );
      HttpWriteAndFree( response, sock );
      SocketShutdown( sock );
      HttpFreeRequest( request );
      sock->data = NULL;
      return;
   }
```

See Also

- HttpFree()

3.21 HttpSetCode()

The function HttpSetCode(Http_t* http, unsigned int code) sets the http code and relevant reason to the Http_t object pointed by the **http**.

Declaration

void HttpSetCode(Http_t* http, unsigned int code)

Parameters

Http_t* http

A pointer to the Http_t object where the code and reason should be set.

unsigned int code

A unsigned int represents the code should be set to the Http_t object.

Returns

This function does not return any values.

See Also

- HttpAddHeader()

3.22 HttpAddHeader()

The function HttpAddHeader(Http_t* http, const char* key, char* value) adds the key and value to the Http_t object header list.

Declaration

void HttpAddHeader(Http_t* http, const char* key, char* value)

Parameters

Http_t* http

A Http_t pointer to the Http_t object where the header should be added.

const char* key

A string pointer to the string represent the key of the data to be added to the Http_t object

char* value

A string pointer represents the value to be added to the Http_t object

Returns

This function does not return any values.

See Also

- HttpAddTextContent()
- HttpSetCode()

3.23 HttpSetContent()

The function HttpSetContent(Http_t* http, char* data, unsigned int length) sets the content passed by the string pointer **data** and sets the size of the content passed by unsigned integer **length** to the Http_t object passed by pointer **http**. It updates the header fields length also.

Declaration

void HttpSetContent(Http_t* http, char* data, unsigned int length)

Parameters

Http_t* http

A pointer to the Http_t object where data to be added.

char* data

A string pointer representing the date should be added.

unsigned int length

A unsigned integer representing the lenght of the data.

Returns

This function does not return any values.

See Also

- HttpAddTextContent()

3.24 HttpAddTextContent()

The function HttpAddTextContent(Http_t* http, char* content) sets the content passed by the string pointer **content** and sets the size of the content to the Http_t object passed by pointer **http**. It updates the header fields length also.

Declaration

void HttpAddTextContent(Http_t* http, char* content)

Parameters

Http_t* http,

A pointer to the Http_t object where data to be added.

char* content

A string pointer representing the content to be added.

Returns

This function does not return any values.

See Also

- HttpSetContent()

3.25 HttpBuild()

The function HttpBuild(Http_t* http) builds string that can represent the Http_t object passed by the pointer **http** and assigned that to the field response of the Http_t object.

Declaration

char* HttpBuild(Http_t* http)

Parameters

Http_t* http

A pointer to the Http_t object.

Returns

It returns the pointer to the string which represent the Http_t object.

See Also

- HttpNew()

3.26 Http400()

The function Http400(Socket_t sock) writes the response "HTTP_400_BAD_REQUEST" to the Socket_t pointed by socket and shut down the socket connection.

Declaration

void Http400(Socket_t* sock)

Parameters

Socket_t* sock

A pointer point to the Socket_t object.

Returns

This function does not return any values.

See Also

- Http400()
- Http403()
- Http404()
- Http500()

3.27 Http403()

The function Http400(Socket_t sock) writes the response "HTTP_403_FORBIDDEN" to the Socket_t pointed by socket and shut down the socket connection.

Declaration

void Http403(Socket_t* sock)

Parameters

Socket_t* sock

This is pointer point to the Socket_t object where the response should be written and shutdown.

Returns

This function does not return any values.

See Also

- Http400()
- Http403()
- Http404()
- Http500()

3.28 Http404()

The function Http404(Socket_t sock) writes the response "HTTP_404_NOT_FOUND" to the Socket_t pointed by socket and shut down the socket connection.

Declaration

void Http404(Socket_t* sock)

Parameters

Socket_t* sock

This is pointer point to the Socket_t object where the response should be written and shutdown.

Returns

This function does not return any values.

See Also

- Http400()
- Http403()
- Http404()
- Http500()

3.29 Http500()

The function Http500(Socket_t sock) writes the response "HTTP_500_INTERNAL_SERVER_ERR
to the Socket_t pointed by socket and shut down the socket connection.

Declaration

void Http500(Socket_t* sock)

Parameters

Socket_t* sock

This is pointer point to the Socket_t object where the response should be written and shutdown.

Returns

This function does not return any values.

See Also

- Http400()
- Http403()
- Http404()
- Http500()

3.30 Httpsprintf()

The function Httpsprintf(char* format, ...)

Declaration

char* Httpsprintf(char* format, ...)

Parameters

char* format

Returns

Examples

See Also

- FileNew()

Chapter 4

List.c

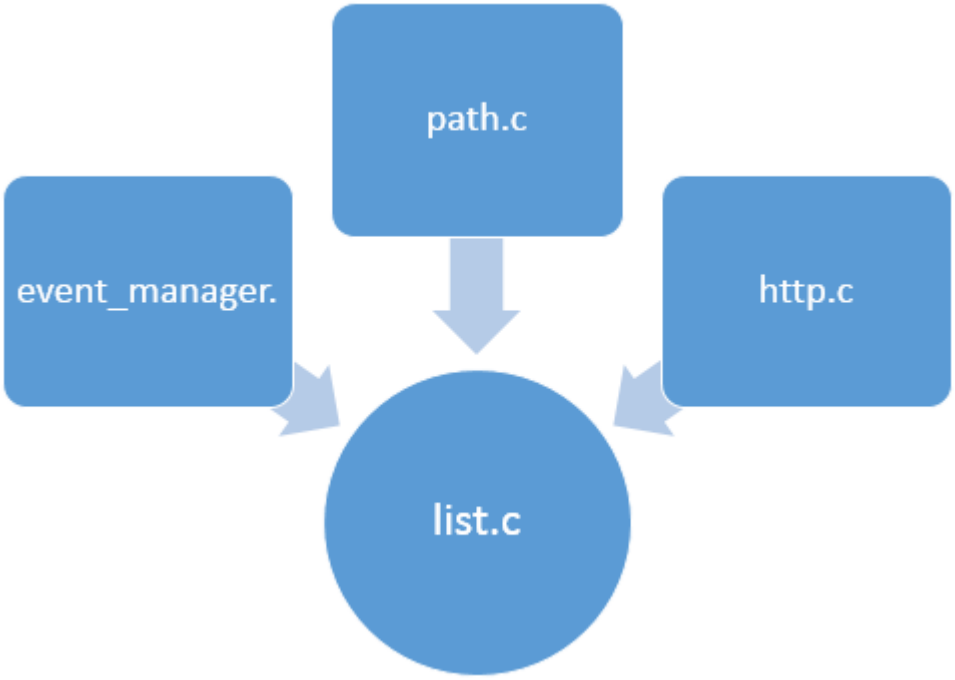

Figure 4.1: Calls to functions of list.c

4.1 CreateList()

The function CreateList() allocates the memory of the size struct List_t and returns the pointer to the allocated memory. It assigns NULL to the fields data and next.

Declaration

List_t* CreateList()

Parameters

This method does not have any input parameters

Returns

This function returns the pointer to the allocated memory.

Examples

```
char* value = "Friend UP";
List_t* list = CreateList(); // Create the List_t struct,
AddToList( list, value ); // Add string "Friend UP" to the List.
```

See Also

- AddToList()
- FreeList()
- ListNew()

4.2 AddToList()

The function AddToList(List_t *list, void *data) adds the passed data pointed ***data** to the List_t object pointed by the pointer **list**. If there is no data exists in the List, the passed data will be added as the first data, else the passed data will be added as the last element of the list.

Declaration

void AddToList(List_t *list, void *data)

Parameters

List_t *list

A pointer to the list where data should be added.

void *data

A pointer to the data.

Returns

The function does not return any values.

Examples

```
char* value = "Friend UP";
List_t* list = CreateList(); // Create the List_t struct,
AddToList( list, value ); // Add string "Friend UP" to the List.
```

See Also

- CreateList()
- FreeList()
- ListAdd()

4.3 FreeList()

The function FreeList(List_t *list) frees the memory of the previously allocated by functions such as CreateList(),ListNew() to create the list pointed by the pointer **list**.

Declaration

void FreeList(List_t *list)

Parameters

List_t *list

A pointer to the list which must be freed.

Returns

This function does not return any values.

Examples

```
char* value = "Friend UP";
char* about = "Friend UP is amazing";
List_t* list = CreateList(); // Create the List_t struct,
AddToList( list, value ); // Add string "Friend UP" to the List.
AddToList( list, about );
FreeList( list ); // the memory used to store the lsit wll be freed
```

See Also

- CreateList()
- AddToList()
- CookiePath()

4.4 ListNew()

The function ListNew() allocated the memory of the size struct List_t and returns the pointer to the allocated memory. It assigns NULL to the fields data and next of the list.

Declaration

List_t* ListNew()

Parameters

This method does not have any parameters.

Returns

This function returns a pointer to the list is created.

Examples

```
char* value = "Friend UP";
List_t* list = ListNew(); // Create the List_t struct,
AddToList( list, value ); // Add string "Friend UP" to the List.
```

See Also

- AddToList()
- FreeList()
- CreateList()

4.5 ListFree()

The function ListFree(List_t *list) deallocate the memory of the previously allocated by functions such as CreateList(),ListNew() to create the list pointed by the pointer **list**.

Declaration

void ListFree(List_t* list)

Parameters

List_t *list

The pointer to the list which must be freed.

Returns

This function does not return any values.

Examples

```
char* value = "Friend UP";
char* about = "Friend UP is amazing";
List_t* list = CreateList(); // Create the List_t struct,
AddToList( list, value ); // Add string "Friend UP" to the List.
AddToList( list, about );
ListFree( list ); // the memory used to store the lsit wll be freed
```

See Also

- CreateList()
- AddToList()
- FreeList()

4.6 ListAdd()

The function ListAdd(List_t *list, void *data) is used to add the supplied data pointed ***data** to the list give by the pointer **list**. If there is no data, the supplied data will be added as the first data, Else the supplied data will be added as the last element of the list. The function returns the list after the append of the data.

Declaration

void ListAdd(List_t *list, void *data)

Parameters

List_t *list

The pointer to the list where data should be added,

void *data

The pointer to the data

Returns

The function returns the pointer to the list where data added.

Examples

```
char* value = "Friend UP";
List_t* list = CreateList(); // Create the List_t struct,
ListAdd( list, value ); // Add string "Friend UP" to the List.
```

See Also

- CreateList()
- FreeList()
- AddToList()

Chapter 5

Path.c

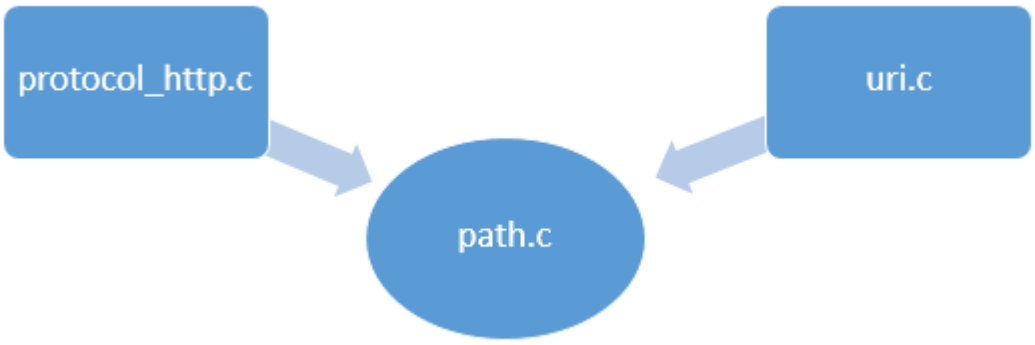

Figure 5.1: Calls to functions of path.c

5.1 PathNew()

The function PathNew(const char* path) allocates the memory in the size of struct Path_t and return the pointer to the allocated memory. The fields of struct Path_t are set to their default values, and field raw is set to the passed string **path** and field raw size set to the size of the string passed. It is used in *protocol_http.c*, *uri.c* classes to create the Path_t struct object from the raw path data.

Declaration

Path_t* PathNew(const char* path)

Parameters

const char* path

A string pointer referring the path, which is used set the fields of the Path_t struct.

Returns

A pointer to created Path_t object is returned.

Example

```
Path_t* path=PathNew( "/friendcore" );
Path_t* base = PathNew( "resources" );
Path_t* complete = PathJoin( base, path );
```

See Also

- PathJoin()
- PathResolve()
- PathMake()
- PathFree()

5.2 PathJoin()

The function PathJoin(Path_t* path1, Path_t* path2) joins two Path_t passed by pointer **path1** and **path2**. A pointer to the combined Path_t is returned. It can be used to combine two part of the path to get the complete path. It is used in the *protocol_http.c* class.

Declaration

Path_t* PathJoin(Path_t* path1, Path_t* path2)

Parameters

Path_t* path1

A pointer to the first Path_t struct which should be joined.

Path_t* path2

A pointer to the second Path_t struct which should be joined.

Returns

It returns A pointer to the combined Path_t struct object.

Example

```
Path_t* path=PathNew( "/friendcore" );
Path_t* base = PathNew( "resources" );
Path_t* complete = PathJoin( base, path );
```

See Also

- PathNew()
- PathResolve()
- PathMake()

5.3 PathSplit()

The function PathSplit(Path_t* p) splits the paths based on separator "/" into array of elements and set the field **parts** of the Path_t struct object. This function used within the *Path.c* class.

Declaration

void PathSplit(Path_t* p)

Parameters

Path_t* p

A pointer to the Path_t which should be split.

Returns

This function does not return any values.

See Also

- PathResolve()
- PathJoin()
- PathMake()

5.4 PathMake()

The function PathMake (Path_t* path) frees the raw path previously set and set the new raw path string to the field **raw**. This function used within the *Path.c* class.

Declaration

void PathMake(Path_t* path)

Parameters

Path_t* path

A pointer to the Path_t where the raw string path should be set.

Returns

This function does not return any values.

Examples

See Also

- PathResolve()
- PathJoin()
- PathNew()

5.5 PathCheckExtension()

The PathCheckExtension(Path_t* path, const char* ext) checks the extension of the Path_t object pointed by pointer **path** with the passed string **ext**. If there is no extension in the given Path_t signed integer **-1** is returned, else an integer value returned based on the comparison.

Declaration

int PathCheckExtension(Path_t* path, const char* ext)

Parameters

Path_t* path

A pointer to the Path_t which the extension should be checked.

const char* ext

A string pointer referring the extension which should be checked with the Path_t object.

Returns

An signed integer value returned based on the comparison. Integer value -1 returned if there is no extension assigned in the Path_t object. Returns 0 if the extension passed and the extension of the path are same.

See Also

- PathResolve()
- PathJoin()
- PathNew()

5.6 PathFree()

The function PathFree(Path_t* path) deallocates the memory which is previously allocated by *PatheNew()* function for the Path_t object pointed out by **path**.

Declaration

void PathFree(Path_t* path)

Parameters

Path_t* path

A pointer to the Path_t object, whose memory should be reallocated.

Returns

This function does not return any values.

See Also

- PathResolve()
- PathJoin()
- PathNew()

5.7 PathResolve()

The function PathResolve(Path_t* p) resolves the paths which was already set. Remove unnecessary characters.

Declaration

void PathResolve(Path_t* p)

Parameters

const char* path

A pointer referring the Path_t object, which should be resolved.

Returns

This function does not return any values.

See Also

- PathSplit()
- PathJoin()
- PathMake()

Chapter 6

File.c

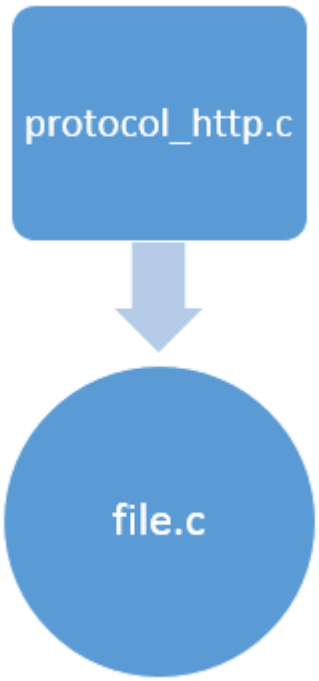

Figure 6.1: Calls to functions of file.c

6.1 FileNew()

The function FileNew(char* path, unsigned int flags) allocates memory in the size of File_t struct. It sets field **path** to the string pointer **path**. It returns the pointer to allocated memory. If the flags was set to FILE_READ_NOW, file pointed by the path will be read and written to the buffer of the File_t struct.

Declaration

File_t* FileNew(char* path, unsigned int flags)

Parameters

char* path

A string pointer referring the path of the file.

unsigned int flags

An unsigned integer referring flag parameter which specifies whether file should be read and the buffer should be written.

Returns

This function returns a pointer to the created File_t struct.

Examples

```
File_t* file = FileNew( "FriendUP.txt", FILE_READ_NOW ); //This
    will create a new File_t struct and the file pointed by the
    path FriendUP.txt will read and writtent to buffer.
```

See Also

- FileRead()
- FileFree()

6.2 FileRead()

The function FileRead(File_t* file, long long offset, long long size) reads a block of memory in the size of **size** and pointed by the **fp** of the struct File_t passed by pointer **file** and the offset given by the **offset**.

Declaration

void FileRead(File_t* file, long long offset, long long size)

Parameters

File_t* file

A pointer to the File_t object which field **fp** to be used as the starting pointer of the memory block to read the file.

long long offset

An integer number referring to the number of bytes to offset from the **fp**.

long long size

An integer number referring the size of the memory block to be read.

Returns

This function does not return any values.

Examples

```
File_t* fo = (File_t*) calloc( 1, sizeof(File_t) );
fo->fp = NULL;
fo->path = NULL;
fo->buffer = NULL;
fo->path = StringDuplicate( path );
FILE* fp = fopen( fo->path, "r+b" );
if( !fp )
{
```

```
    fo->exists = false;
    return fo;
  }
  fo->exists = true;
  fo->fp = fp;
  fseek( fp, 0L, SEEK_END );
  fo->filesize = ftell( fp );
  if( flags & FILE_READ_NOW )
  {
    FileRead( fo, 0, fo->filesize );
  }
```

See Also

- FileNew()
- FileFree()

6.3 FileFree()

The function FileFree(File_t* file) deallocates the memory which is previously allocated by FileNew() function for the File_t object pointed out by pointer **file**. If a null pointer is passed as argument, no action occurs.

Declaration

void FileFree(File_t* file)

Parameters

File_t* file

A pointer to the File_t object, whose memory should be reallocated.

Returns

This function does not return any value.

Examples

```
File_t* file = FileNew( "FriendUP.txt", FILE_READ_NOW );
FileFree(file);
```

See Also

- FileNew()
- FileRead()

Chapter 7

uri.c

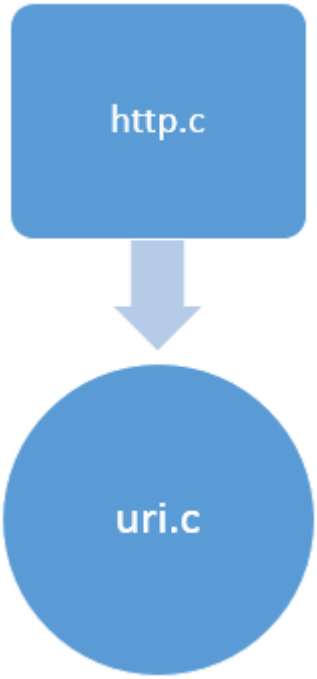

Figure 7.1: Calls to functions of uri.c

7.1 UriNew()

The function UriNew() allocates the memory in the size of Uri_t and returns the pointer to the allocated memory.

Declaration

Uri_t* UriNew()

Parameters

This function does not have input parameters.

Returns

This function returns a pointer to the allocated memory.

See Also

- UriFree()

7.2 UriParseQuery()

The function UriParseQuery(char* query) parses the query into a list of key, value pairs and put the pairs into a Hashmap_t and returns a pointer to the Hashmap_t object created.

Declaration

Hashmap_t* UriParseQuery(char* query)

Parameters

char* path

A string pointer passed, which representing the query to be parsed.

Returns

A pointer to the Hashmap_t returned.

See Also

- UriParse()

7.3 UriParse()

The function UriParse(char* str) parses the string passed and allocates memory with the size of struct Uri_t and set the fields of the Uri_t struct. This function returns a Uri_t pointer to the memory allocated.

Declaration

Uri_t* UriParse(char* str)

Parameters

char* str

A string pointer representing the string to be parsed.

Returns

A pointer to the Uri_t returned.

Examples

```
char* const uristring="http://someone@example.org/noewhere.txt";
Uri_t* uri=UriParse(uri);
printf("Scheme: %s\n", uri->scheme);
//above code print follows
//Scheme is http
```

See Also

- UriParseQuery()

7.4 UriFree()

The function UriFree deallocated the memory which was previously allocated store the Uri_t object pointed by pointer **uri**. It also deallocates the memory allocated to store the Authority_t, Path, Scheme, Query, fragment, queryRaw.

Declaration

void UriFree(Uri_t* uri);

Parameters

Uri_t* uri

This is a pointer to the Uri_t object, whose memory should be reallocated.

Returns

This function does not return any values.

See Also

- UriNew()

Chapter 8

websocket.c

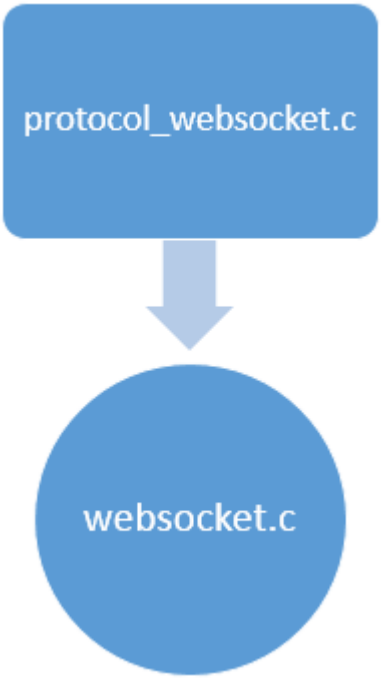

Figure 8.1: Calls to functions of websocket.c

8.1 WebsocketNew()

The function WebsocketNew() allocates the memory with size of struct Webscoket_t and returns a pointer to allocated memory. The fields of struct Webscoket_t are set to their default values (NULL).

Declaration

Websocket_t* WebsocketNew()

Parameters

This function does not have input parameters.

Returns

This function returns a pointer to the allocated memory to store the Websocket_t object and returns NULL if request failed because of lack of memory.

See Also

- WebsocketFree()

8.2 WebsocketAccept()

The function WebsocketAccept(Socket_t* sock, Http_t* request) accepts a websocket connection. The function returns true if websocket connection was accepted or returns false if the connection was failed. If the connection was accepted, the response is written to the Socket pointed by **sock**. The connection request should be HTTP version **V1.1**, request type **GET** and websocket version should be **v13**.

Declaration

bool WebsocketAccept(Socket_t* sock, Http_t* request)

Parameters

Socket_t* sock

A pointer to the Socket_t where the connection should be accepted and response should be written.

Http_t* request

A pointer to the Http_t request.

Returns

This function returns true if connection was accepted or returns false if connection was rejected or failed.

See Also

- WebsocketParsePartial()

8.3 WebsocketParsePartial()

The function WebsocketParsePartial() parses the header passed by Websocker_t pointer.

Declaration

int WebsocketParsePartial(Websocket_t* ws, char* data, unsigned int length)

Parameters

Websocket_t* ws

The pointer to the Websocket_t header of which should be parsed.

char* data

unsigned int length

Returns

The function returns integer which represent the state of the parsing.

See Also

- WebsocketAccept()

8.4 WebsocketBuild()

The function WebsocketBuild() allocates a memory block of given size and set the raw data passed by string pointer **raw** returns a pointer to the allocated memory block.

Declaration

char* WebsocketBuild(char* raw, unsigned int size, unsigned int* messageSize)

Parameters

char* raw

A string pointer referring the date to be written

unsigned int size

An unsgned integer referring the size of the memory should be allocated

unsigned int* messageSize

A unsigned integer used to set the size of the message.

Returns

This function returns the pointer to memory where the string was saved.

See Also

- WebsocketAccept()

8.5 WebsocketFree()

The function WebsocketFree() deallocates the memory which was previously allocated by WebsocketNew() to save the Websocket_t object.

Declaration

void WebsocketFree(Websocket_t* ws)

Parameters

Websocket_t* ws

A pointer to the Websocket_t object, whose memory should be reallocated.

Returns

This function does not return any values.

See Also

- WebsocketNew()

Chapter 9

hashmap.c

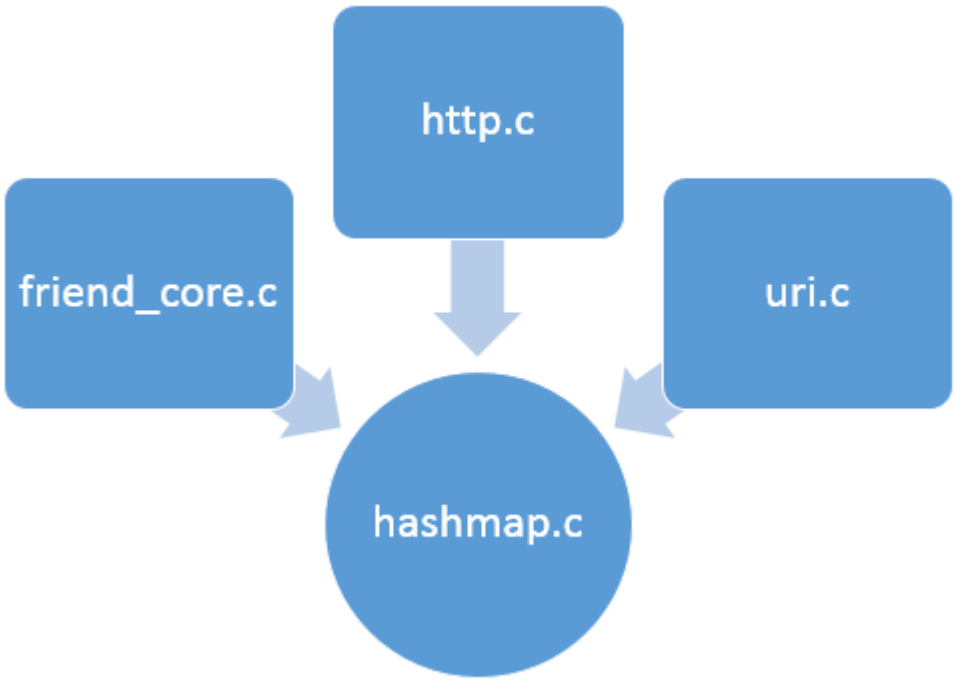

Figure 9.1: Calls to functions of hashmap.c

9.1 HashmapNew()

The function HashmapNew() allocates memory with the size of Hashmap_t and returns the pointer to the allocated memory to store the Hashmap_t.

Declaration

Hashmap_t* HashmapNew()

Parameters

This function does not take any input parameters.

Returns

This function return the pointer to the allocated memory to store the Hashmap_t.

See Also

- HashmapFree()

9.2 HashmapPut()

The function HashmapPut(Hashmap_t* in, char* key, void* value) adds the value to the Hashmap_t pointed by **in** with the key **key**. If the Hashmap_t pointed by **in** full, Hashmap_t will be rehash and the key and value will be added. It returns true on success and false on failure.

Declaration

bool HashmapPut(Hashmap_t* in, char* key, void* value)

Parameters

Hashmap_t* in

A pointer to the Hashmap_t where they key and value to be added.

char* key

A string pointer to the key associated with the value to be added

void* value

A pointer the the value to be added to the Hashmap_t.

Returns

This function return true on success and false on failure.

Examples

See Also

- HashmapGet()

9.3 HashmapGet()

The function HashmapGet(Hashmap_t* in, char* key) get the element pointed by **key** from the Hashmap_t pointer by **in** and return a pointer to the HashmapElement_t on successful. Returns null if element was not found.

Declaration

HashmapElement_t* HashmapGet(Hashmap_t* in, char* key)

Parameters

Hashmap_t* in

A pointer to the Hashmap_t object, where the pointer to the value should be found for the key.

char* key

A string pointer referring the key.

Returns

This function returns pointer to the HashmapElement_t associated with the key passed on success, returns null if element was not found.

See Also

- HashmapPut()

9.4 HashmapIterate()

The function HashmapIterate(Hashmap_t* in, unsigned int* iterator) iterates.

Declaration

HashmapElement_t* HashmapIterate(Hashmap_t* in, unsigned int* iterator)

Parameters

Hashmap_t* in

unsigned int* iterator

Returns

See Also

- HashmapPut()

9.5 HashmapFree()

The function HashampFree deallocates the memory which is previously allocated by function **HashmapNew** to store the Hashmap_t object. It does

deallocates the memory to store the pointer to the HashmapElement_t. It does not deallocates the memory to store the HashmapElement_t.

Declaration

void HashmapFree(Hashmap_t* in)

Parameters

Hashmap_t* in

This is the pointer to a memory block to be deallocated which was previously allocated by **HashmapNew**. If a null pointer is passed as argument, no action occurs.

Returns

This function does not return any values.

See Also

- HashmapNew()

9.6 HashmapLength()

This function finds the length of the Hasmap_t object passed by the pointer **in**.

Declaration

int HashmapLength(Hashmap_t* in)

Parameters

Hashmap_t* in

It is pointer to the Hashmap_t object which length should be found.

Returns

Returns the length of the Hashmap_t object. If NULL pointer passed it returns 0.

See Also

- HashmapGet()

Chapter 10

base64.c

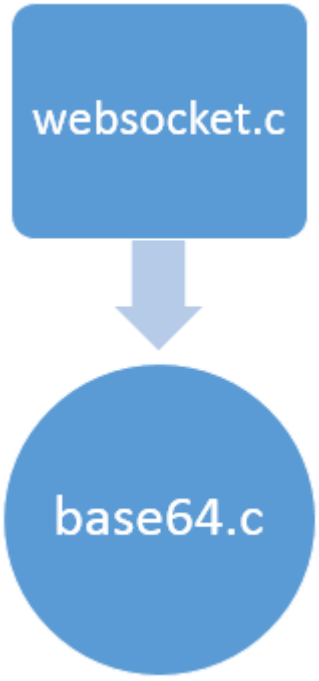

Figure 10.1: Calls to functions of base64.c

10.1 Base64Encode()

This function encode the string passed by string pointer **data**.

Declaration

char* Base64Encode(const unsigned char* data, int length)

Parameters

const unsigned char* data

A string pointer to the data which should be encoded.

int length

An integer representing the length of the of the string to be encoded.

Returns

It returns a pointer to the encoded string.

See Also

- Base64Decode()

10.2 Base64Decode()

This function decode the string which was previously encoded.

Declaration

char* Base64Decode(const unsigned char* data, int length)

Parameters

const unsigned char* data

A string pointer to the data which should be decoded.

int length

An integer representing the length of the of the string to be decoded.

Returns

It returns a pointer to the decoded string.

See Also

- Base64Encode()

Chapter 11

protocol_http.c

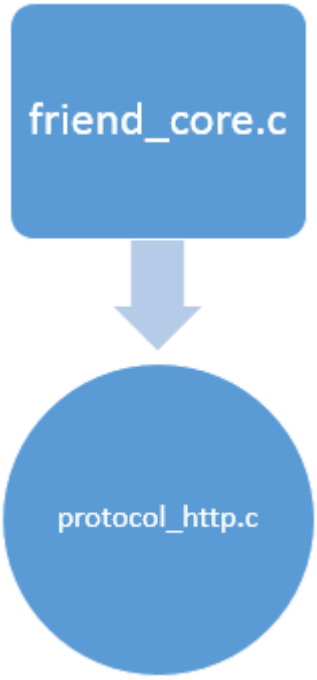

Figure 11.1: Calls to functions of protocol_http.c

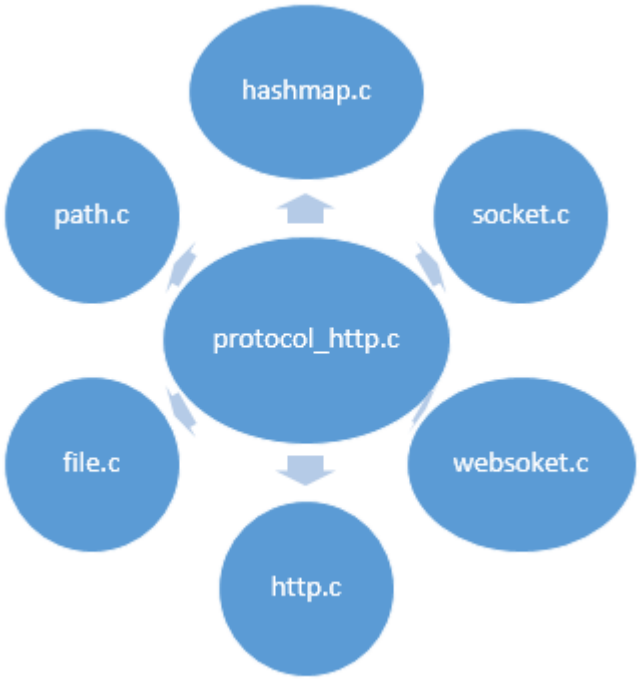

Figure 11.2: Calls from protocol_http.c

11.1 ProtocolHttp()

Declaration

void ProtocolHttp(Socket_t* sock, char* data, unsigned int length)

Parameters

Socket_t* sock

char* data

unsigned int length

Returns

This function does not return any values.

Examples

Chapter 12

event_manager.c

12.1 EventManagerNew()

It allocates the memory with the size of struct **EventManager** and return the pointer to the allocated memory. It set's the last id of the EventManager object to 0*f and new list object will be created and the pointer to the list will be assigned to the eventList field.

Declaration

EventManager *EventManagerNew()

Parameters

This function does not have any input parameters.

Returns

It returns the pointer to the allocated memory which was used to store the **EventManager** object.

See Also

- EventManagerDelete()

12.2 EventManagerDelete()

It deallocates the memory which was previously allocated to store the **EventManager** object pointed by the pointer **em**.

Declaration

void EventManagerDelete(EventManager *em)

Parameters

EventManager *em

A pointer the EventManager object, whose memory should be reallocated.

Returns

This function does not return any values.

See Also

- EventManagerNew()

12.3 EventGetNewID()

This function get the last ID and returns the ID which could be used as new ID for the event.

Declaration

ULONG EventGetNewID(EventManager *em)

Parameters

EventManager *em

A pointer to the EventManager to get the new ID type of ULONG for the new event.

Returns

This function returns event ID type of ULONG.

See Also

- EventManagerNew()

12.4 EventAdd()

This function add the event to the end of the list if there exist a list of same event, else new list will be created and even will be added.

Declaration

CoreEvent *EventAdd(EventManager *em, ULONG id, ULONG *h_Function)

Parameters

EventManager *em

A pointer to the EventManaget where the new event should be added.

ULONG id

This represent the id of the event to be created.

ULONG *h_Function

Returns

This function return the pointer to the created event.

See Also

- EventCheck()

12.5 EventCheck()

This function EventCheck(EventManager *em, ULONG id) find and return the event with the field **id** same as **if** in the event list attached to the EventManager pointed by **em**.

Declaration

CoreEvent *EventCheck(EventManager *em, ULONG id)

Parameters

EventManager *em

A pointer to the EventManaget where the event should be found.

ULONG id

This represent the id of the event to be found.

Returns

This function return the pointer to the created event.

See Also

- EventAdd()

Chapter 13

class.c

13.1 ClassCreate()

The function ClassCreate(ClassID cid, Class *rc, struct Hook *disp) allocates the memory in the size of struct **class** to save the object type of class struct. The class member **cl_ID** set as **cid** and member super class **cl_Super** set to the class pointed by **rc**, other members are set to default.

Declaration

Class *ClassCreate(ClassID cid, Class *rc, struct Hook *disp)

Parameters

ClassID cid

Class *rc

Pointer to struct Class representing the super class.

struct Hook *disp

Returns

A pointer to a Class struct if the class was created. If the library cannot be created or an error occurred, a NULL pointer will be returned.

See Also

- HashmapNew()

13.2 ClassDelete()

The function ULONG ClassDelete(struct Class *c) deallocates the memory previously allocated by the ClassCreate() function, if the class object pointed by pointer **c** have some object still attached to it or Subclass attached to it the function will not deallocates the memory.

Declaration

ULONG ClassDelete(struct Class *c)

Parameters

struct Class *c

Pointer to Class object to be released.

Returns

It returns ULON 0 if it successful, else return 1 if the class object pointed by pointer **c** have some object still attached , else returns 2 if the class pointed by pointer **c** have some Subclass objects attached.

See Also

- HashmapNew()

13.3 ObjectNewF()

Declaration

Object *ObjectNewF(Class *c, Object *o, struct Msg *msg)

Parameters

Socket_t* sock

char* data

unsigned int length

Returns

This function does not return any values.

See Also

- HashmapNew()

13.4 ObjectDelete()

Declaration

void ObjectDelete(Object *o)

Parameters

Object *o

Returns

This function does not return any values.

See Also

- HashmapNew()

13.5 DoSuperMethod()

Declaration

ULONG DoSuperMethod(Class *c, Object *o, struct Msg *msg)

Parameters

Class *c

Object *o

struct Msg *msg

Returns

See Also

- HashmapNew()

Chapter 14

Cookie.c

14.1 CookieNew()

The function CookieNew(char* name, char* value) allocates the memory with the size of the Cookie_t struct and returns the pointer to the allocated memory. It sets the Cookie_t field **name** and **value** with the value passed. If **name** or **value** is null or the memory could not be allocated, the function returns null.

Declaration

Cookie_t* CookieNew(char* name, char* value)

Parameters

char* name

A string pointer to the name of the cookie to be created.

char* value

A string pointer to the value of the cookie.

Returns

It returns a pointer to the allocated memory where the cookie is saved.

See Also

- CookieExpires()
- CookiePath()

14.2 CookieExpires()

The function CookieExpires(Cookie_t* cookie, time_t date) sets the expire time to the Cookie which is pointed by a pointer **cookie**.

Declaration

CookieExpires(Cookie_t* cookie, time_t date)

Parameters

Cookie_t* cookie

A pointer to the Cookie_t where the expire time should be set.

time_t date

The expired time of the cookie

Returns

The function does not returns any values.

See Also

- CookieNew()
- CookiePath()

14.3 CookiePath()

The function CookiePath(Cookie_t* cookie, char* path) sets the path passed by string pointer **path** to the Cookie_t passed by pointer **cookie**.

Declaration

void CookiePath(Cookie_t* cookie, char* path)

Parameters

Cookie_t* cookie

A pointer to the Cookie_t where path should be set.

char* path

A string pointer to the path, which should be set to the cookie.

Returns

The function does not returns any values.

See Also

- CookieExpires()
- CookieNew()
- CookieDomain()

14.4 CookieDomain()

The function CookieDomain(Cookie_t* cookie, char* domain) sets the domain passed by string pointer **domain** to the Cookie_t pointed by **cookie**.

Declaration

void CookieDomain(Cookie_t* cookie, char* domain)

Parameters

Cookie_t* cookie

A pointer to the Cookie_t where domain should be set.

char* domain

A string pointer to the domain which should be set to the cookie.

Returns

This function does not returns any values.

See Also

- CookieExpires()
- CookieNew()
- CookiePath()

14.5 CookieSecure()

The function CookieSecure(Cookie_t* cookie, int secure) sets the field **secure** to make sure the cookie passed by the pointer **cookie** only transmitted through the HTTPS or not.

Declaration

void CookieSecure(Cookie_t* cookie, int secure)

Parameters

Cookie_t* cookie

A pointer to the cookie where the field **secure** should be set to make sure the cookie transmitted through HTTPS

int secure

An integer value 0 or 1, based the value the cookie will be transmitted through HTTPS or not.

Returns

This function does not return any values.

See Also

- CookieExpires()
- CookieNew()
- CookiePath()

14.6 CookieHttpOnly()

The function CookieHttpOnly(Cookie_t* cookie, int httpOnly) set the field **httpOnly** of struct Cookie_t in order to make sure that the cookie should be available through http(s) only.

Declaration

void CookieHttpOnly(Cookie_t* cookie, int httpOnly)

Parameters

Cookie_t* cookie

A pointer to the cookie where the field **httpOnly** should be set to make sure the cookie transmitted through http.

int httpOnly

An integer value 0 or 1, based the value the cookie will be transmitted through HTTP(s) or not.

Returns

This function does not return any values

See Also

- CookieExpires()
- CookieNew()
- CookiePath()

14.7 CookieMake()

The function CookieMake(Cookie_t* cookie) creates the string which represent the Cookie_t object passed by pointer **cookie**. A pointer to the created string returned.

Declaration

char* CookieMake(Cookie_t* cookie)

Parameters

Cookie_t* cookie

A pointer to the Cookie_t struct, which should be created as string.

Returns

This function returns a string pointer which represents the created string.

See Also

- CookieExpires()
- CookieNew()
- CookiePath()

14.8 CookieFree()

The function CookieFree(Cookie_t* cookie) deallocates the memory previously allocated to store the Cookie_t object pointed by the pointer **cookie**.

Declaration

void CookieFree(Cookie_t* cookie)

Parameters

Cookie_t* cookie

A pointer to the cookie which memory to be reallocated.

Returns

This function does not returns any values.

See Also

- CookieExpires()
- CookieNew()
- CookiePath()

Chapter 15

Socket.c

15.1 classes calls functions of socket.c

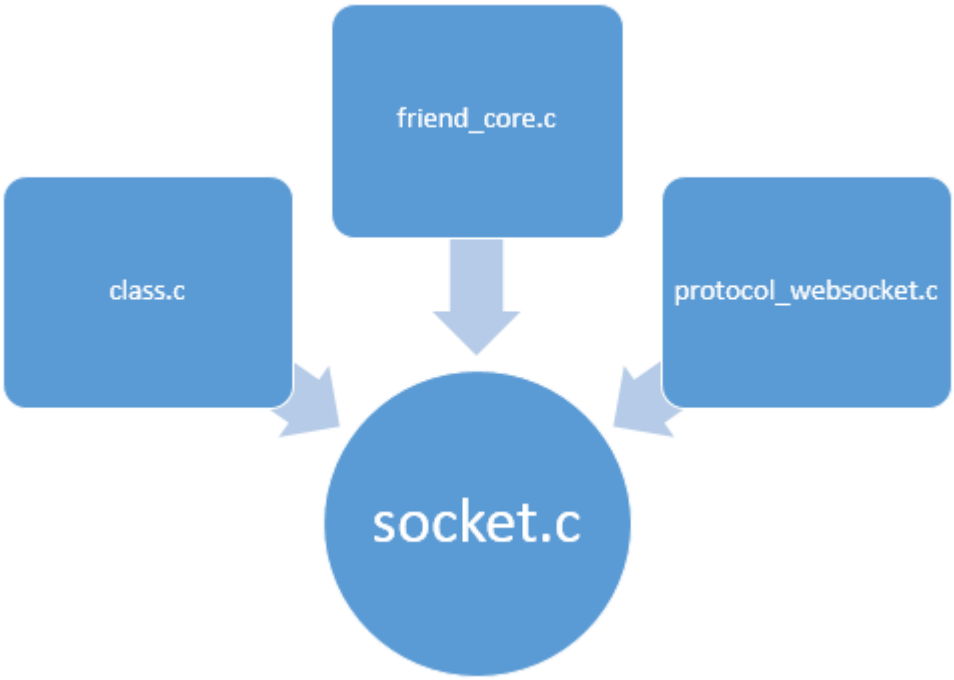

Figure 15.1: Calls to functions of Socket.c

15.2 SocketOpen()

The function SocketOpen(unsigned short port) is used to create a Socket connection on the specified port and return the pointer to the opened socket, returns false if open a connection failed.

Declaration

Socket_t* SocketOpen(unsigned short port)

Parameters

unsigned short port

unsigned short is supplied which represent the port to be opened.

Returns

The function return the pointer to the opened socket.

See Also

- SocketListen()
- SocketSetBlocking()
- SocketAccept()

15.3 SocketListen()

The function SocketListen(Socket_t* sock) used to make the socket **sock** listen for incoming connection. Returns true on success and false on failure.

Declaration

int SocketListen(Socket_t* sock)

Parameters

Socket_t* sock

The pointer to the Socket_t is given which should be listen to incoming connections.

Returns

Returns true on success and false on failure.

See Also

- SocketOpen()
- SocketSetBlocking()
- SocketAccept()

15.4 SocketSetBlocking()

The function SocketSetBlocking(Socket_t* sock, bool block) is used to set the socket pointed by the pointer **sock** to blocking or non-blocking based on the bool value **block**. It returns true on success and false on failure.

Declaration

int SocketSetBlocking(Socket_t* sock, bool block)

Parameters

Socket_t* sock

The pointer to the Socker_t which must be set blocking or non-blocking.

bool block

The bool valued supplied, based on the valued, the supplied Socket_t pointed by sock will be set blocking or non-blocking.

Returns

Returns true on success and false on failure.

See Also

- SocketOpen()
- SocketWrite()
- SocketAccept()

15.5 SocketAccept()

The function SocketAccept(Socket_t* sock) is used to accept the incoming connection and return a new Socket_t object if connection was accepted. Return NULL if connection was refused or error occurred.

Declaration

Socket_t* SocketAccept(Socket_t* sock)

Parameters

Socket_t *list

The pointer to the Socket_t which should accept the connection.

Returns

A new Socket_t object if connection was accepted. Return NULL if connection was refused or error occurred.

See Also

- SocketOpen()
- SocketSetBlocking()
- SocketAccept()

15.6 SocketWrite()

The function int SocketWrite(Socket_t* sock, char* data, unsigned int length) is used to write the pointed by the pointer **data** with the supplied **length** to the Socket_t pointer by the pointer **sock**. The function returns true on success and false on failure.

Declaration

int SocketWrite(Socket_t* sock, char* data, unsigned int length)

Parameters

Socket_t *list

The pointer to the Socket_t where data must be written.

char* data

The pointer to the data which must be written.

unsigned int length

The length of the data should be written.

Returns

The function returns true on success and false on failure.

See Also

- SocketOpen()
- SocketSetBlocking()
- SocketAccept()

15.7 SocketShutdown()

The function void SocketShutdown(Socket_t* sock) is used to shutdown the Socket_t pointed by the pointer **sock**.

Declaration

void SocketShutdown(Socket_t* sock)

Parameters

Socket_t *sock

The pointer to the Socket_t which must be shutdown.

Returns

The function does not return any values.

See Also

- SocketOpen()
- SocketSetBlocking()
- SocketClose()

15.8 SocketClose()

The function void SocketClose(Socket_t* sock) is used to forcely close the Socket_t pointed by the pointer **sock** and free the memory.

Declaration

void SocketClose(Socket_t* sock)

Parameters

Socket_t *sock

The pointer to the Socket_t which must be closed and free the memory.

Returns

The function does not return any values.

Examples

```
printf("Open a Socket connection.\n");
Socket_t* newsocket = SocketOpen( 6502 );

printf("Make the socket listned.\n");
if( SocketListen( newsocket )
{
   printf("Set socket non-blocking.\n");
   if(SocketSetBlocking( newsocket, 0)
   {
      printf("Set socket accept.\n");
      if(SocketAccept( newsocket , 1 )
      {
         printf("Write Data to socket.\n");
         unsigned int length = 0;
         char* response = "Friend UP is AWSOME";
         if( SocketWrite( sock, response, length );)
         {
            SocketShutdown ( sock );
         }else
         {
            SocketClose( sock );
         }

      }
   }
```

See Also

- SocketOpen()
- SocketSetBlocking()
- SocketShutdown()

Chapter 16

buffered_string.c

16.1 BufStringInit()

The function BufStringInit() allocates the memory in the size of struct **BufString** and it allocates the buffer and assign the pointer to the buffer property of the **BufString** struct, and return the pointer to the allocated memory. It returns NULL if request failed because of no memory.

Declaration

BufString *BufStringInit()

Parameters

This function does not take any input parameters.

Returns

This function returns the pointer to the allocated memory if request successful, else it return NULL.

See Also

- BufStringDeInit()

16.2 BufStringDeInit()

The function BufStringDeInit deallocates the memory passed by pointer **bs** which was previously allocated by function **BufStringInit()**.

Declaration

void BufStringDeInit(BufString *bs)

Parameters

BufString *bs

This is pointer to the struct **BugString** which should be freed.

Returns

This function does not return any values.

See Also

- BufStringInit()

16.3 BufStringAdd()

The function BufStringAdd(BufString *bs, const char * ntext) writes the string **ntext** to the **BufString** struct object and set the struct property **size**. If the size of the buffer is not enough to hold the string **ntext**, the buffer will be extended.

Declaration

void BufStringAdd(BufString *bs, const char *ntext)

Parameters

BufString *bs

This is the pointer to the BufString where the string passed should be added.

const char *ntext

This is the string to be added to the BufString struct object.

Returns

This function does not return any values.

See Also

- BufStringDeInit()

Chapter 17

mainclass.c

17.1 EventAdd()

Declaration

CoreEvent *EventAdd(EventManager *em, ULONG id, ULONG *h_Function)

Parameters

Socket_t* sock

char* data

unsigned int length

Returns

This function does not return any values.

See Also

- HashmapNew()

Chapter 18

Service.c

18.1 GetSuffix()

The function GetSuffix returns the pointer to the constant suffix **appache**.

Declaration

const char *GetSuffix()

Parameters

This function doesn't take any parameters.

Returns

It returns the pointer to the constant suffix **apache**.

Examples

```
char* suffix=GetSuffix();
printf("%s", suffix);
// About code snippet print as follows
// apache
```

See Also

- ServiceOpen()

18.2 ServiceOpen()

The function ServiceOpen is open the service passed by the name and version.

Declaration

struct Service* ServiceOpen(char* name, long version)

Parameters

char* name

This is the name of the service to opened.

long version

This is the version of the service to opened.

Returns

It returns the pointer to the service opened, It returns NULL if error occurred or couldn't find the service.

See Also

- GetSuffix()

18.3 ServiceClose()

This function closed the service which previously opened by call to function **ServiceOpen** and free the memory allocated to hold the service object.

Declaration

void ServiceClose(struct Service* service)

Parameters

struct Service* service

This is the pointer to the service object to be closed and freed.

Returns

This function does not return any values.

Examples

```
char* servicename="";
Service *locserv = ServiceOpen( servicename, 0 ); // open the
    Service pointed by servicename and version
if(!locserv==NULL)
{
ServiceClose(locserv); // close the service
}
```

See Also

- ServiceOpen()
- ServiceDelete()

18.4 GetVersion()

The function GetVersion(void) is returns the ULONG version.

Declaration

ULONG GetVersion(void)

Parameters

void

This function does not take any values.

Returns

This function returns the ULONG version.

Examples

```
ULONG version = GetVersion();
printf("%s", version);
//Above function print the following
// 1
```

See Also

- GetRevision()

18.5 GetRevision()

The function GetRevision() returns the ULON revision.

Declaration

ULONG GetRevision(void)

Parameters

This function does not take any values.

Returns

This function returns the ULONG revision number.

Examples

See Also

- GetVersion()

18.6 ServiceNew()

The function ServiceNew(STRPTR command) allocates the memory to hold the service object and returns the pointer to the created service object.

Declaration

Service *ServiceNew(STRPTR command)

Parameters

STRPTR command

Returns

This function returns the pointer to created service object.

Examples

See Also

- ServiceOpen()

18.7 ServiceDelete()

The function ServiceDelete(Service *s) deallocates the memory previously allocated by the function **ServiceNew**.

Declaration

void ServiceDelete(Service *s)

Parameters

Service *s

This is pointer to the Service object to be freed.

Returns

This function does not return any values.

Examples

See Also

- ServiceNew()

18.8 ServiceStart()

The function ServiceStart(Service *service) start the service passes by the pointer **service** to the service object. It returns 0 if the request failed, els it returns 1.

Declaration

int ServiceStart(Service *service)

Parameters

Service *service

This is pointer to the service to be started.

Returns

This function returns 0 if the request failed, els it returns 1.

Examples

See Also

- ServiceNew()
- ServiceStop()

18.9 ServiceStop()

The function ServiceStop(Service *s) stop the service passed by the service pointer **s**. It returns 0 if the request failed, els it returns 1.

Declaration

int ServiceStop(Service *s)

Parameters

Service *s

This is pointer to the Service to be stopped.

Returns

This function returns 0 if the request failed, els it returns 1.

Examples

See Also

- ServiceStart()

18.10 ServiceThread()

Declaration

int ServiceThread(FThread *ptr)

Parameters

FThread *ptr

Returns

Examples

See Also

- ServiceStart()

Chapter 19

service_manager.c

19.1 ServiceManagerNew()

The function ServiceManagerNew() allocates memory in the size of Service-Manager and returns the pointer to the allocated memory. It returns NULL on failure.

Declaration

ServiceManager *ServiceManagerNew()

Parameters

This function does not take any parameters.

Returns

See Also

- ServiceManagerDelete()

19.2 ServiceManagerDelete()

The function ServiceManagerDelete() closes all the services attached to the service manager and deallocates the memory previously allocated by call the function ServiceManagerNew().

Declaration

void ServiceManagerDelete(ServiceManager *smgr)

Parameters

ServiceManager *smgr

This is the pointer to the ServiceManager to be deleted.

Returns

This function doesn't return any values.

See Also

- ServiceManagerNew()

19.3 ServiceManagerGetByName()

The function ServiceManagerGetByName(ServiceManager *smgr, STRPTR name) get the service with the name passed by **name name** from the service manager passed by pointer **smgr**. It returns NULL if n service found.

Declaration

Service *ServiceManagerGetByName(ServiceManager *smgr, STRPTR name)

Parameters

ServiceManager *smgr

This is pointer to the ServiceManager where the service to be found.

STRPTR name

This is the name of the service to be found.

Returns

This function returns the pointer to the Service if found, else returns NULL.

See Also

- ServiceManagerNew()

19.4 ServiceManagerChangeServiceState()

The function ServiceManagerChangeServiceState(ServiceManager *smgr, Service *srv, int state) change the state of the Service passed by pointer **srv** which is attached to the ServiceManager passed by pointed **smgr** to the state passed by the integer **state**.

Declaration

int ServiceManagerChangeServiceState(ServiceManager *smgr, Service *srv, int state)

Parameters

ServiceManager *smgr

This is pointer to the ServiceManager where the service to be found.

Service *srv

This is pointer to the Service which state should be changed.

int state

This is the integer represent the state.

Returns

It returns 0 on failure or any other integer on success.

See Also

- ServiceManagerNew()

Chapter 20

friend_core.c

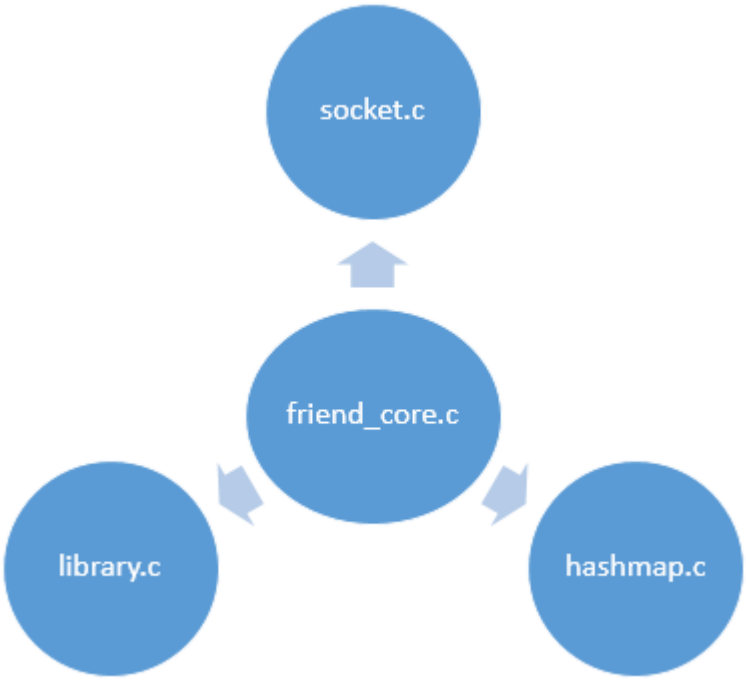

Figure 20.1: Calls from friend_core.c

20.1 FriendCoreNew()

The function FriendCoreNew() allocates the memory in the size of FriendCoreInstance_t struct and return the pointer to the allocated memory. It allocated memory in the size of ServiceManager struct and assign the pointer to the ServiceManager to the servicemanager property of the FriendCoreInstance_t struct. It returns NULL if it can't allocate the memory. There is not enough memory is the reason for the failure.

Declaration

FriendCoreInstance_t *FriendCoreNew()

Parameters

This function does not take any input parameters.

Returns

This function returns the pointer to the allocated memory if request successful, returns NULL if request fails.

See Also

- FriendCoreShutdown()

20.2 FriendCoreShutdown()

The function FriendCoreShutdown() closes all sockets, signals shutdown to all subsystems FriendCoreRun will return shortly after this.

Declaration

void FriendCoreShutdown(FriendCoreInstance_t* instance);

Parameters

FriendCoreInstance_t* instance

This is the pointer to the FriendCoreInstance_t to be shut down.

Returns

This function does not return any values.

See Also

- FriendCoreNew()

20.3 FriendCoreRun()

The function ServiceInstall() uninstall the Apache service.

Declaration

int FriendCoreRun(FriendCoreInstance_t* instance)

Parameters

ApacheService *s

This is the pointer to the AppacheService to be uninstalled.

Returns

Examples

See Also

- ServiceInstall()

20.4 FriendCoreEpoll()

The function FriendCoreEpoll()

Declaration

void FriendCoreEpoll(FriendCoreInstance_t* instance)

Parameters

FriendCoreInstance_t* instance

This is the pointer to the AppacheService to be uninstalled.

Returns

This function doesn't return any values.

See Also

- FriendCoreRun()

20.5 FriendCoreGetLibrary()

The function FriendCoreGetLibrary(FriendCoreInstance_t* instance, char* libname, long version) get the Library with the name pointed by *libname* and version passed by *version* from the FrientCoreInstance_t passed by pointer *instance* if available, Else It open the library and assign the library pointer to the FriendCoreInstance_t libraries property and return the pointer to the library.

Declaration

Library_t* FriendCoreGetLibrary(FriendCoreInstance_t* instance, char* libname, long version)

Parameters

FriendCoreInstance_t* instance

This is the pointer to the FriendCoreInstance_t where libraries to be found.

char* libname

The pointer to the name of the library.

long version

This is the version of the library.

Returns

It returns the pointer to the library if found in the FriendCoreInstance_t object or if it could open the library passed by the name and version, Else ir returns NULL.

See Also

- ServiceInstall()

20.6 ServiceUninstall()

The function ServiceInstall() uninstall the Apache service.

Declaration

int ServiceUninstall(ApacheService *s)

Parameters

ApacheService *s

This is the pointer to the AppacheService to be uninstalled.

Returns

Examples

See Also

- ServiceInstall()

Chapter 21

friendcore_manager

21.1 FriendCoreManagerNew()

The function FriendCoreManagerNew() allocates the memory in the size of FriendCoreManager struct and if it successful, it allocates memory in the size of FriendCoreInstance_t and assign the pointer to the assign the pointer to property fcm_FriendCores. If memory couldn't allocated to structs FriendCoreManager or FriendCoreInstance_t, it returns NULL.

Declaration

FriendCoreManager *FriendCoreManagerNew()

Parameters

This function does not take any parameters.

Returns

This function return the pointer to the created FriendCoreManager object.

See Also

- FriendCoreManagerDelete()

21.2 FriendCoreManagerDelete()

Declaration

void FriendCoreManagerDelete(FriendCoreManager *fcm)

Parameters

ApacheService *s

This is the pointer to the AppacheService to be uninstalled.

Returns

Examples

See Also

- FriendCoreManagerNew()

21.3 FriendCoreManagerRunCores()

Declaration

ULONG FriendCoreManagerRunCores(FriendCoreManager *fcm)

Parameters

FriendCoreManager *fcm

This is the pointer to the FriendCoreManager struct.

Returns

See Also

- FriendCoreManagerNew()

21.4 FriendCoreManagerNew()

Declaration

ULONG FriendCoreManagerRunCores(FriendCoreManager *fcm)

Parameters

ApacheService *s

This is the pointer to the AppacheService to be uninstalled.

Returns

Examples

See Also

- ServiceInstall()

Chapter 22

cAjax.js

22.1 open()

The function open() Specifies the type of request, the URL, and if the request should be handled asynchronously or not and whether is has return code.

Declaration

function(method, url, syncing, hasReturnCode)

Parameters

method

Type of the method (GET or POST) is passed by this parameter. If nothing passed, http method **POST** assigned. Else the passed value is changed to upper case.

url

The location of the file on the server

syncing

Boolean value to specify whether it is sync method or not.

hasReturnCode

Boolean value to specify whether is has a return code.

Returns

This function doesn't return anyvalues.

See Also

- send()
- addVar()

22.2 addVar()

The function addVar() adds the variable with the **key** and **value** to the Ajax query.

Declaration

function(key, val)

Parameters

key

The name of the variable which should be added to the Ajax query.

val

The value of the variable which should be added to the Ajax query.

Returns

This function returns boolean true.

See Also

- send()
- setRequestHeader()

22.3 setRequestHeader()

The function setRequestHeader()Sets the value of an HTTP request header. You must call setRequestHeader()after open(), but before send().

Declaration

function(type, data)

Parameters

type

The name of the header whose value is to be set.

data

The value to set as the body of the header.

Returns

This function returns boolean true.

See Also

- send()
- open()

22.4 send()

Sends the request. If the method type is **POST**, this function sends the data passed if it is not null, else it sends the variables. If the method type is

GET,this function sends null. If the request is asynchronous (which is the default), this method returns as soon as the request is sent. If the request is synchronous, this method does not return until the response has arrived.

Declaration

function(data)

Parameters

data

The data to be sent. If the data is empty, variables will be sent.

Returns

If the request is asynchronous (which is the default), this method returns as soon as the request is sent. If the request is synchronous, this method does not return until the response has arrived.

See Also

- open()
- addVar()
- setRequestHeader()

22.5 responseText()

The function responseText return the response for the request.

Declaration

function()

Parameters

This function does not have any input parameters.

Returns

This function return the response text.

See Also

- open()
- addVar()
- setRequestHeader()

Chapter 23

door.js

23.1 Door()

The function Door() sets the property path, and set the handler property to "void" and call the init() function.

Declaration

function(path)

Parameters

path

The path to be set to path property of the Door.

Returns

This function does not return any values.

23.2 get()

The function get() return **false** if the was null, else it compares the path element with the volume of the icons, if they matches, it returns the Door object associated with the matched icon, else return the new Door object with the path set.

Declaration

function(path)

Parameters

path

The path to be set to path property of the Door.

Returns

This function does not return any values.

See Also

- Door()

23.3 setPath()

The function setPath() sets the path property of the Door object and it does not return any values.

Declaration

function(path)

Parameters

path

The path to be set to path property of the Door.

Returns

This function does not return any values.

See Also

- get()

23.4 dosAction()

Declaration

function(func, args, callback)

Parameters

func

The path to be set to path property of the Door.

args

callback

Returns

This function does not return any values.

See Also

- get()

23.5 getIcons()

Declaration

function(fileInfo, callback)

Parameters

fileInfo

The path to be set to path property of the Door.

callback

Returns

This function does not return any values.

See Also

- get()

Chapter 24

csssparcer.js

24.1 ParseCssFile()

The function ParseCssFile() parse String/ file and make a compact logical css.

Declaration

ParseCssFile (string, path)

Parameters

string

The string which should be parsed.

path

The path to the file which should be parsed.

Returns

This function does not return any values.

See Also

- AddParsedCSS()

24.2 AddParsedCSS()

The function ParseCssFile() parse String/ file and make a compact logical css.

Declaration

ParseCssFile (string, path)

Parameters

string

The string which should be parsed.

path

The path to the file which should be parsed.

Returns

This function does not return any values.

See Also

- get()