PEter

Generated by Doxygen 1.8.13

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

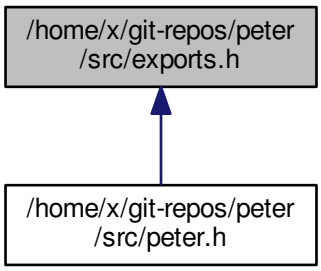
Chapter 2

File Documentation

2.1 /home/x/git-repos/peter/src/exports.h File Reference

These functions can be used to get information from a PE files export header.

This graph shows which files directly or indirectly include this file:



Functions

- uint16_t exports_get_ordinal_by_name (void *image, const char *name)
  returns the ordinal for the function you name
- void * exports_get_function_by_name (void *image, const char *name)
  returns a functions address
- char * exports_get_name_by_index (void *image, const uint32_t idx)
  returns the first function name in the export headerif idx = 0, the second if idx = 1 and so on
- void * exports_get_function_by_ordinal (void *image, uint32_t ord)
  you give it an ordinal and you get a pointer to a function

2.1.1 Detailed Description

These functions can be used to get information from a PE files export header.

2.1.2 Function Documentation

2.1.2.1 exports_get_function_by_name()

```
void* exports_get_function_by_name (
            void * image,
            const char * name )
```

returns a functions address

Parameters

image	is a pointer to the pe image you want to analyze
name	is a pointer on the name of the API function you want the address of

2.1.2.2 exports_get_function_by_ordinal()

```
void* exports_get_function_by_ordinal (
            void * image,
            uint32_t ord )
```

you give it an ordinal and you get a pointer to a function

Parameters

image	<– like above
ord	should be one of the values you can find in the odinal list of the PEs export table.

2.1.2.3 exports_get_name_by_index()

```
char* exports_get_name_by_index (
            void * image,
            const uint32_t idx )
```

returns the first function name in the export headerif idx = 0, the second if idx = 1 and so on

Parameters

image	is a pointer on the pe image
idx	is 0 for the first name and so on... (like an array index)

2.1.2.4 exports_get_ordinal_by_name()

```
uint16_t exports_get_ordinal_by_name (
            void * image,
            const char * name )
```

returns the ordinal for the function you name

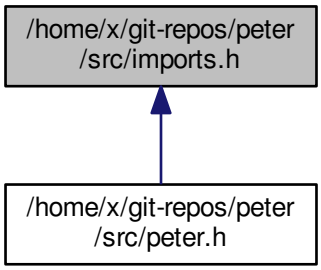
Parameters

image	is a pointer to the first byte of the image you want to analyze
name	is a pointer on the name of the function you want to get the ordinal of

2.2 /home/x/git-repos/peter/src/imports.h File Reference

These functions give you information about the import table.

This graph shows which files directly or indirectly include this file:



Functions

- IMAGE_IMPORT_DESCRIPTOR * imports_by_index_get_import_descriptor (void *image, int i)

  return the import descriptor with the index i
- char * imports_by_index_get_name (void *image, int i)

  *returns a char *pointer on the import descriptors name (the dll name)*
- int imports_by_name_get_index (void *image, const char *name)

  returns the import descroptor index that fits the name parameter
- int imports_by_name_get_iname_index_by_name (void *image, const char *dllName, const char *funcName)

  returns the index of the imported fucntion of a dll (that's not an ordinal)
- IMAGE_IMPORT_BY_NAME * imports_by_index_get_iname_by_index (void *image, int idtIdx, int intIdx)

  you give it two indexes (import descriptor and its import name table) and you get the name of the function these two indexes discribe. That means you will get the first function name of the first imported dll if idtIdx and intIdx are both == 0. The returned pointer is an IMAGE_IMPORT_BY_NAME struct which is actually a uint8_t array but with a word (16bit) value at its start.

- void * imports_by_index_get_iaddr_thunk_by_index (void *image, int idtIdx, int iatIdx)

  like imports_by_index_get_iname_by_index() but doesn't return a name but a pointer on IMAGE_THUNK_DATA if it's a 32bit executable this function will return IMAGE_THUNK_DATA32 and if you have 64bit you may use the return value as a pointer on IMAGE_THUNK_DATA64

2.2.1 Detailed Description

These functions give you information about the import table.

2.2.2 Function Documentation

2.2.2.1 imports_by_index_get_iaddr_thunk_by_index()

```
void* imports_by_index_get_iaddr_thunk_by_index (
            void * image,
            int idtIdx,
            int iatIdx )
```

like imports_by_index_get_iname_by_index() but doesn't return a name but a pointer on IMAGE_THUNK_DATA if it's a 32bit executable this function will return IMAGE_THUNK_DATA32 and if you have 64bit you may use the return value as a pointer on IMAGE_THUNK_DATA64

Parameters

image	is the PE file
idtIdx	is an import descriptor index
iatIdx	is a name table index

2.2.2.2 imports_by_index_get_import_descriptor()

```
IMAGE_IMPORT_DESCRIPTOR* imports_by_index_get_import_descriptor (
            void * image,
            int idtIdx )
```

return the import descriptor with the index i

Parameters

image	points to the image you want to analyze
i	is 0 if you need the first import descriptor i = 1 for the second and so on

2.2.2.3 imports_by_index_get_iname_by_index()

```
IMAGE_IMPORT_BY_NAME* imports_by_index_get_iname_by_index (
            void * image,
            int idtIdx,
            int intIdx )
```

you give it two indexes (import descriptor and its import name table) and you get the name of the function these two indexes discribe. That means you will get the first function name of the first imported dll if idtIdx and intIdx are both == 0. The returned pointer is an IMAGE_IMPORT_BY_NAME struct which is actually a uint8_t array but with a word (16bit) value at its start.

Parameters

image	points to the PE file
idtIdx	is the import descriptor index (starting at 0 for the first import descriptor)
intIdx	is the name table index (also starting with 0)

2.2.2.4 imports_by_index_get_name()

```
char* imports_by_index_get_name (
            void * image,
            int idtIdx )
```

returns a char ∗pointer on the import descriptors name (the dll name)

Parameters

image	points tor the image you want to analyzee
i	is an index indicating which import disc. you want the name of (starting with 0)

2.2.2.5 imports_by_name_get_iname_index_by_name()

```
int imports_by_name_get_iname_index_by_name (
            void * image,
            const char * dllName,
            const char * funcName )
```

returns the index of the imported fucntion of a dll (that's not an ordinal)

Parameters

image	points to the PE file you want information of
dllName	is the name of the dll (import descriptor name) where the function name resides in
funcName	is the name of the function you want the index of

2.2.2.6 imports_by_name_get_index()

```
int imports_by_name_get_index (
            void * image,
            const char * name )
```

returns the import descroptor index that fits the name parameter

Parameters

image	points on the image you want to analyze
name	the dll name (== import descriptor name) you want the idx of

2.3 /home/x/git-repos/peter/src/peter.h File Reference

This file is the file you can actually include for using the library. It does nothing but including other header files.

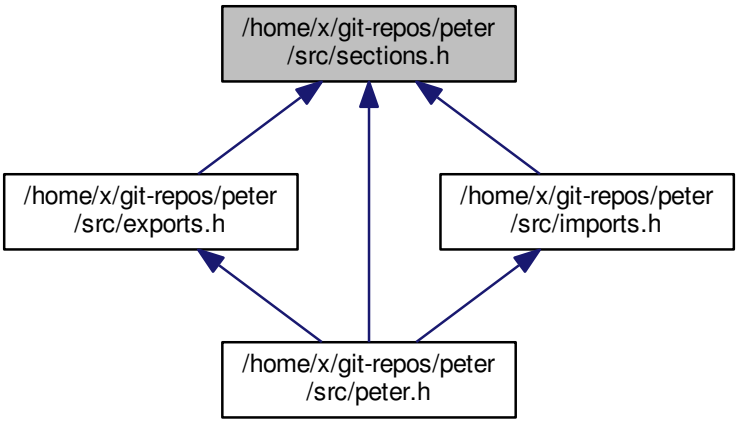
2.3.1 Detailed Description

This file is the file you can actually include for using the library. It does nothing but including other header files.

2.4 /home/x/git-repos/peter/src/sections.h File Reference

This file contains functions for getting information concerning the section headers (e.g. sections_by_name_is_writable() informs you wether the writable flag in a section header is set). An other type of function changes the values within the section headers (set functions like sections_by_name_set_name()) or may even add a new one. The functions which convert e.g. pointers to rva values where placed in this header too, because they need information from the section headers to calculate their return values.

This graph shows which files directly or indirectly include this file:



Functions

- IMAGE_SECTION_HEADER * sections_by_index_get_sh (void *image, int index)

  returns a pointer on the IMAGE_SECTION_HEADER at the indexed poisition.
- IMAGE_SECTION_HEADER * sections_by_name_get_sh (void *image, const char *name)

  like sections_by_index_get_sh() but index is replaced by the section name.
- int sections_by_name_get_index (void *image, const char *name)

  Tells you at which position the section header with the name xy lies.
- uint32_t sections_by_name_get_characteristics (void *image, const char *name)

  returns the characteristics field of a section header.
- int sections_by_name_is_writable (void *image, const char *name)

  returns 1 if the section is writable and 0 if not
- int sections_by_name_is_readable (void *image, const char *name)

  returns 1 if section is readable and 0 if not
- int sections_by_name_is_executable (void *image, const char *name)

  returns 1 if the section is executable and 0 if not.
- void * sections_by_name_get_pointer_to_memory (void *image, const char *name)

  returns a pointer on the first byte of a section
- uint32_t rva_to_raw (char *image, uint32_t rva)

  you give it a 32bit rva and it returns the raw offset
- void * rva_to_ptr (void *image, uint32_t rva)
- uint32_t raw_to_rva (char *image, uint32_t raw)

  returns a rva if you give it a raw offset
- uint32_t ptr_to_rva (char *image, char *ptr)

  makes a rva address from a pointer
- void sections_by_name_set_writable (void *image, const char *name)

  Sets the writable flag in the section header.
- void sections_by_name_set_readable (void *image, const char *name)

  sets the readable flag in the section header
- void sections_by_name_set_executable (void *image, const char *name)

sets the executable flag in the section header

- void sections_by_name_set_name (void *image, const char *name, const char *replace)

  replaces the section name with a section name of your choise (8 byte max)
- void sections_by_name_set_characteristics (void *image, const char *name, uint32_t characteristics)

  sets the characteristics field of a section header.
- void * sections_add_section (void *image, int imageSz, void *spaceRetImage, int spaceSz, const char *name, uint32_t size)

  Creates a new PE image from image. The new image will be written to spaceSpaceRetImage is there is enough space. The new image will be like the old but with a section added to it. You can calculate the number of bytes you must allocate with: sections_get_sizeof_image_after_add_section(). The return value is a pointer on the first byte of the new section.
- uint32_t sections_get_sizeof_image_after_add_section (void *oldImage, uint32_t oldSz, uint32_t addedSz)

  calculates the size, the image will have after adding a section
- void * sections_by_name_enlarge (void *image, const char *name, char *bytes, uint32_t bytesLen)

  enlarges a section and returns a pointer on the first byte of the new allocated space. If it fails it returns NULL.

2.4.1 Detailed Description

This file contains functions for getting information concerning the section headers (e.g. sections_by_name_is_writable() informs you wether the writable flag in a section header is set). An other type of function changes the values within the section headers (set functions like sections_by_name_set_name()) or may even add a new one. The functions which convert e.g. pointers to rva values where placed in this header too, because they need information from the section headers to calculate their return values.

2.4.2 Function Documentation

2.4.2.1 ptr_to_rva()

```
uint32_t ptr_to_rva (
            char * image,
            char * ptr )
```

makes a rva address from a pointer

Parameters

image	
ptr	

2.4.2.2 raw_to_rva()

```
uint32_t raw_to_rva (
            char * image,
            uint32_t raw )
```

returns a rva if you give it a raw offset

Parameters

image	
raw	

2.4.2.3 rva_to_ptr()

```
void* rva_to_ptr (
            void * image,
            uint32_t rva )
```

Parameters

image	
rva	

2.4.2.4 rva_to_raw()

```
uint32_t rva_to_raw (
            char * image,
            uint32_t rva )
```

you give it a 32bit rva and it returns the raw offset

Parameters

image	is the PE file.
rva	

2.4.2.5 sections_add_section()

```
void* sections_add_section (
            void * image,
            int imageSz,
            void * spaceRetImage,
            int spaceSz,
            const char * name,
            uint32_t size )
```

Creates a new PE image from image. The new image will be written to spaceSpaceRetImage is there is enough space. The new image will be like the old but with a section added to it. You can calculate the number of bytes you must allocate with: sections_get_sizeof_image_after_add_section(). The return value is a pointer on the first byte of the new section.

Parameters

image	is the old pe image
imageSz	is the size of the old image in bytes
spaceRetImage	should point on an allocated space in memory
spaceSz	must contain the size in bytes of the memory spaceRetImage is pointing to.
name	is the name of the new section (not longer than 8 bytes)
size	is how big you want the new section to be (in bytes)

2.4.2.6 sections_by_index_get_sh()

```
IMAGE_SECTION_HEADER* sections_by_index_get_sh (
            void * image,
            int index )
```

returns a pointer on the IMAGE_SECTION_HEADER at the indexed poisition.

Parameters

image	is a pointer on the PE file in memory.
index	starts with 0 and is 1 if you want the second section header.

2.4.2.7 sections_by_name_enlarge()

```
void* sections_by_name_enlarge (
            void * image,
            const char * name,
            char * bytes,
            uint32_t bytesLen )
```

enlarges a section and returns a pointer on the first byte of the new allocated space. If it fails it returns NULL.

Parameters

image	is the PE image
name	is the name of the section you want to enlarge
bytes	are the bytes you want to add to the section. It can be NULL.
bytesLen	is the size in bytes by which you want the section to be enlarged.

2.4.2.8 sections_by_name_get_characteristics()

```
uint32_t sections_by_name_get_characteristics (
```

```
            void * image,
            const char * name )
```

returns the characteristics field of a section header.

Parameters

image	is like always the PE file.
name	is the section name.

2.4.2.9 sections_by_name_get_index()

```
int sections_by_name_get_index (
            void * image,
            const char * name )
```

Tells you at which position the section header with the name xy lies.

Parameters

image	is the PE file.
name	is the name of the section (max 8 byte)

2.4.2.10 sections_by_name_get_pointer_to_memory()

```
void* sections_by_name_get_pointer_to_memory (
            void * image,
            const char * name )
```

returns a pointer on the first byte of a section

Parameters

image	is the PE file.
name	is the section name.

2.4.2.11 sections_by_name_get_sh()

```
IMAGE_SECTION_HEADER* sections_by_name_get_sh (
            void * image,
            const char * name )
```

like sections_by_index_get_sh() but index is replaced by the section name.

Parameters

image	is the PE file.
name	is a pointer on the section name. It should not exceed 8 bytes since the section names of PE files are not alowed to be longer than 8 bytes.

2.4.2.12 sections_by_name_is_executable()

```
int sections_by_name_is_executable (
            void * image,
            const char * name )
```

returns 1 if the section is executable and 0 if not.

Parameters

image	is the PE file.
name	is the section name.

2.4.2.13 sections_by_name_is_readable()

```
int sections_by_name_is_readable (
            void * image,
            const char * name )
```

returns 1 if section is readable and 0 if not

Parameters

image	is the PE file.
name	is the section name.

2.4.2.14 sections_by_name_is_writable()

```
int sections_by_name_is_writable (
            void * image,
            const char * name )
```

returns 1 if the section is writable and 0 if not

Parameters

image	is the PE file.
name	is the section name

2.4.2.15 sections_by_name_set_characteristics()

```
void sections_by_name_set_characteristics (
            void * image,
            const char * name,
            uint32_t characteristics )
```

sets the characteristics field of a section header.

Parameters

image	is the PE file.
name	is the name of the section
characteristics	is the value the characteristics filed should be changed to.

2.4.2.16 sections_by_name_set_executable()

```
void sections_by_name_set_executable (
            void * image,
            const char * name )
```

sets the executable flag in the section header

Parameters

image	
name	

2.4.2.17 sections_by_name_set_name()

```
void sections_by_name_set_name (
            void * image,
            const char * name,
            const char * replace )
```

replaces the section name with a section name of your choise (8 byte max)

Parameters

image	is the PE file
name	is the section name you want to replace
replace	is the new section name

2.4.2.18 sections_by_name_set_readable()

```
void sections_by_name_set_readable (
            void * image,
            const char * name )
```

sets the readable flag in the section header

Parameters

image	
name	

2.4.2.19 sections_by_name_set_writable()

```
void sections_by_name_set_writable (
            void * image,
            const char * name )
```

Sets the writable flag in the section header.

Parameters

image	is the PE image
name	is the section name

2.4.2.20 sections_get_sizeof_image_after_add_section()

```
uint32_t sections_get_sizeof_image_after_add_section (
            void * oldImage,
            uint32_t oldSz,
            uint32_t addedSz )
```

calculates the size, the image will have after adding a section

Parameters

oldImage	is the PE image before adding a section
oldSz	is the size of the image, oldImage is pointing to
addedSz	is the size of the section you want to add.

Index