NAME

caesar_standard – the ‘‘standard’’ library of OPEN/CAESAR

PURPOSE

The ‘‘standard’’ library provides basic types and notations shared by all *OPEN/CAESAR* modules.

USAGE

The ‘‘standard’’ library only consists of a predefined header file **`caesar_standard.h`**.

FEATURES

....

```
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include <signal.h>
```

Various standard C libraries are imported, notably those providing functions for dynamic memory allocation, input/output, byte string facilities, and software signals. In particular, the **`FILE`** type and file handling primitives are imported.

....

`CAESAR_TYPE_NATURAL`

```
typedef unsigned long CAESAR_TYPE_NATURAL;
```

`CAESAR_TYPE_NATURAL` is the unsigned integer type used in *OPEN/CAESAR*.

....

`CAESAR_TYPE_INTEGER`

```
typedef long CAESAR_TYPE_INTEGER;
```

`CAESAR_TYPE_INTEGER` is the signed integer type used in *OPEN/CAESAR*.

....

`CAESAR_TYPE_BOOLEAN`

```
typedef unsigned char CAESAR_TYPE_BOOLEAN;
```

`CAESAR_TYPE_BOOLEAN` is the boolean type used in *OPEN/CAESAR*. It follows the usual conventions of the C language: 0 means ‘‘false’’ and any value different from 0 means ‘‘true’’.

....

`CAESAR_FALSE`

```
#define CAESAR_FALSE ((CAESAR_TYPE_BOOLEAN) 0)
```

`CAESAR_FALSE` is the ‘‘false’’ boolean value.

....

`CAESAR_TRUE`

```
#define CAESAR_TRUE ((CAESAR_TYPE_BOOLEAN) 1)
```

`CAESAR_TRUE` is one possible ‘‘true’’ boolean value.

....

`CAESAR_TYPE_BYTE`

```
typedef unsigned char CAESAR_TYPE_BYTE;
```

`CAESAR_TYPE_BYTE` is the byte type used in *OPEN/CAESAR*.

....

`CAESAR_TYPE_STRING`

```
typedef char *CAESAR_TYPE_STRING;
```

`CAESAR_TYPE_STRING` is the string type (pointer to a character string termined by a null character) used in *OPEN/CAESAR*.

....

`CAESAR_TYPE_FILE`

```
typedef FILE *CAESAR_TYPE_FILE;
```

`CAESAR_TYPE_FILE` is the file type (pointer to a POSIX file descriptor) used in *OPEN/CAESAR*.

....

`CAESAR_TYPE_POINTER`

```
typedef unsigned char *CAESAR_TYPE_POINTER;
```

`CAESAR_TYPE_POINTER` is a pointer to a (generic) byte string.

....

`CAESAR_TYPE_GENUINE_INT`

```
typedef int CAESAR_TYPE_GENUINE_INT;
```

`CAESAR_TYPE_GENUINE_INT` is a 32-bit integer type; its usage is discouraged, except at specific places where the C˜standard explicitly requires an ‘‘int’’ type to be used.

....

CAESAR_TYPE_ARGC

```
typedef CAESAR_TYPE_GENUINE_INT CAESAR_TYPE_ARGC;
```

CAESAR_TYPE_ARGC is a 32-bit integer type; it is used to declare the **argc** parameter of the **main()** function of a C program.

....

CAESAR_TYPE_ARGV

```
typedef char **CAESAR_TYPE_ARGV;
```

CAESAR_TYPE_ARGV is a pointer to an array of strings; it is used to declare the **argv** parameter of the **main()** function of a C program.

....

CAESAR_SIZE_POINTER

```
CAESAR_TYPE_NATURAL CAESAR_SIZE_POINTER ()
   { ... }
```

This function returns the size (in bytes) of a value of type **CAESAR_TYPE_POINTER**.

....

CAESAR_ALIGNMENT_POINTER

```
CAESAR_TYPE_NATURAL CAESAR_ALIGNMENT_POINTER ()
   { ... }
```

This function returns the alignment factor (in bytes) for a value of type **CAESAR_TYPE_POINTER**.

Note: The alignment factor is often dependent from the machine architecture. For any memory area (and not only those of type **CAESAR_TYPE_POINTER**), the alignment factor is always a power of two (1, 2, 4, ...) and is an exact divider of the size of the area. Whatever the machine architecture, any memory area must start at a machine address that is an even multiple of the alignment factor for that area, this constraint stating that areas must be properly aligned on machine word boundaries. In general, one has only to care about alignment constraints when creating structures containing areas with different alignment factors.

....

CAESAR_TYPE_GENERIC_FUNCTION

```
typedef void (*CAESAR_TYPE_GENERIC_FUNCTION) ();
```

CAESAR_TYPE_GENERIC_FUNCTION is the “pointer to a function” type used in *OPEN/CAESAR*. The number of parameters for this function, the types of these parameters (if any) and the result type for this function are not specified.

....

CAESAR_TYPE_COMPARE_FUNCTION

```
typedef CAESAR_TYPE_BOOLEAN (*CAESAR_TYPE_COMPARE_FUNCTION)
   (CAESAR_TYPE_POINTER, CAESAR_TYPE_POINTER);
```

CAESAR_TYPE_COMPARE_FUNCTION is the "pointer to a comparison function" type used in *OPEN/CAESAR*. A comparison function takes two parameters (two pointers to data of the same type) and returns a boolean value, depending whether both pointers refer to identical data or not.

....

CAESAR_TYPE_HASH_FUNCTION

```
typedef CAESAR_TYPE_NATURAL (*CAESAR_TYPE_HASH_FUNCTION)
   (CAESAR_TYPE_POINTER, CAESAR_TYPE_NATURAL);
```

CAESAR_TYPE_HASH_FUNCTION is the "pointer to a hash function" type used in *OPEN/CAESAR*. A hash function takes two parameters (a pointer to data and a natural number N) and returns a natural number in the range 0..N-1.

....

CAESAR_TYPE_CONVERT_FUNCTION

```
typedef CAESAR_TYPE_STRING (*CAESAR_TYPE_CONVERT_FUNCTION)
   (CAESAR_TYPE_POINTER);
```

CAESAR_TYPE_HASH_FUNCTION is the "pointer to a conversion function" type used in *OPEN/CAESAR*. A conversion function takes one parameter (a pointer to data) and returns a character string containing the data under some printable form.

....

CAESAR_TYPE_PRINT_FUNCTION

```
typedef void (*CAESAR_TYPE_PRINT_FUNCTION)
   (CAESAR_TYPE_FILE, CAESAR_TYPE_POINTER);
```

CAESAR_TYPE_PRINT_FUNCTION is the "pointer to a printing procedure" type used in *OPEN/CAESAR*. A printing procedure takes two parameters (a pointer to a file and a pointer to data) and prints the latter to the former.

....

CAESAR_TYPE_FORMAT

```
typedef unsigned char CAESAR_TYPE_FORMAT;
```

CAESAR_TYPE_FORMAT is the type used to control the output of the various printing functions (i.e., functions whose name starts with **CAESAR_PRINT_**) specified in the *OPEN/CAESAR* "graph module" interface or provided by the *OPEN/CAESAR* library.

Many of these functions can display a given object under various formats with different degree of information (e.g., a concise format vs a verbose format).

Each format is represented by a small value of type **CAESAR_TYPE_FORMAT**, e.g., 0, 1, 2, etc.

The format to be used by a printing function is not passed to this function as an argument (this would be too heavy); instead, the format is stored either in a global variable or in a field contained in the object to be printed.

....

CAESAR_INVALID_FORMAT

```
#define CAESAR_INVALID_FORMAT ((CAESAR_TYPE_FORMAT) 255)
```

CAESAR_INVALID_FORMAT is a special value of the type **CAESAR_TYPE_FORMAT**. It can be returned as a result by function **CAESAR_HANDLE_FORMAT()** when this function is invoked with an incorrect argument.

....

CAESAR_CURRENT_FORMAT

```
#define CAESAR_CURRENT_FORMAT ((CAESAR_TYPE_FORMAT) 254)
```

CAESAR_CURRENT_FORMAT is a special value of the type **CAESAR_TYPE_FORMAT**. It can be passed as an argument to function **CAESAR_HANDLE_FORMAT()** and to the various functions whose name starts with **CAESAR_FORMAT_** so as to query the current value of the global variable or object field that stores the format used by the corresponding printing function.

....

CAESAR_MAXIMAL_FORMAT

```
#define CAESAR_MAXIMAL_FORMAT ((CAESAR_TYPE_FORMAT) 253)
```

CAESAR_MAXIMAL_FORMAT is a special value of the type **CAESAR_TYPE_FORMAT**. It can be passed as an argument to function **CAESAR_HANDLE_FORMAT()** and to the various functions whose name starts with **CAESAR_FORMAT_** so as to query the largest format value supported by the corresponding printing function.

....

CAESAR_PRINT_FORMAT

```
void CAESAR_PRINT_FORMAT (CAESAR_FILE, CAESAR_FORMAT)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_FORMAT CAESAR_FORMAT;
   { ... }
```

This procedure prints to file **CAESAR_FILE** the value of **CAESAR_FORMAT**. If **CAESAR_FORMAT** is equal to one of the three special values **CAESAR_INVALID_FORMAT**, **CAESAR_CURRENT_FORMAT**, or **CAESAR_MAXIMAL_FORMAT**, its value is printed symbolically rather than numerically.

Before this procedure is called, **CAESAR_FILE** must have been properly opened, for instance using

fopen(3).

....

CAESAR_HANDLE_FORMAT

```
CAESAR_TYPE_FORMAT CAESAR_HANDLE_FORMAT (CAESAR_VARIABLE, CAESAR_VALUE,
                                         CAESAR_MAXIMAL_VALUE)
   CAESAR_TYPE_FORMAT *CAESAR_VARIABLE;
   CAESAR_TYPE_FORMAT CAESAR_VALUE;
   CAESAR_TYPE_FORMAT CAESAR_MAXIMAL_VALUE;
   { ... }
```

This function is an auxiliary function that should be used to define all the various functions whose name starts with **CAESAR_FORMAT_**. Examples of the proper use of this function can be found by inspecting the C˜code generated by *CAESAR* or *EXP.OPEN*.

This function consults or modifies the format value pointed to by **CAESAR_VARIABLE**, which is expected to be the address of the global variable or object field that stores the format used by the corresponding printing function.

If **CAESAR_VALUE** is equal to **CAESAR_CURRENT_FORMAT**, this function returns the current value of the format pointed to by **CAESAR_VARIABLE**.

If **CAESAR_VALUE** is equal to **CAESAR_MAXIMAL_FORMAT**, this function returns **CAESAR_MAXIMAL_VALUE**, which is expected to be the largest format value supported by the corresponding printing function.

If **CAESAR_VALUE** is less or equal to **CAESAR_MAXIMAL_VALUE**, then the value of the format pointed to by **CAESAR_VARIABLE** is modified and set to **CAESAR_VALUE**. In such case, this function returns an undefined value.

In all other cases, this function returns **CAESAR_INVALID_FORMAT**.

....

CAESAR_TYPE_ABSTRACT

```
#define CAESAR_TYPE_ABSTRACT(CAESAR_NAME) struct CAESAR_NAME *
```

CAESAR_TYPE_ABSTRACT (CAESAR_NAME) is a pointer to a structure of name **CAESAR_NAME**.

Usually, the definition of **CAESAR_NAME** is not available, so that **CAESAR_TYPE_ABSTRACT (...)** is a “generic” pointer to some data structure, the internal representation of which is not pertinent to the user. This data structure is “abstract”, in the sense that one should not rely on a particular implementation.

If **CAESAR_X** is an expression of type **CAESAR_TYPE_ABSTRACT (...)**, dereferencing **CAESAR_X** (e.g., ***CAESAR_X** or **CAESAR_X->...**) is not allowed. Note: if **CAESAR_X** is an expression of type **CAESAR_TYPE_ABSTRACT (...)**, its value usually obeys alignment constraints (for instance, it can be an address multiple of 4).

....

CAESAR_CREATE

```
#define CAESAR_CREATE(CAESAR_ADDRESS,CAESAR_SIZE,CAESAR_TYPE) \
        (CAESAR_ADDRESS) = (CAESAR_TYPE) malloc (CAESAR_SIZE)
```

This macro-definition encapsulates the C function **malloc(3)**. It allocates a memory area of **CAESAR_SIZE** bytes and assigns its address to variable **CAESAR_ADDRESS**, which should be of type **CAESAR_TYPE**. If allocation fails, a **NULL** pointer is assigned to **CAESAR_ADDRESS**.

....

CAESAR_DELETE

```
#define CAESAR_DELETE(CAESAR_ADDRESS) \
        free ((char *) (CAESAR_ADDRESS))
```

This macro-definition encapsulates the C function **free(3)**. It frees the memory area pointed to by **CAESAR_ADDRESS**.

....

CAESAR_TOOL

```
extern CAESAR_TYPE_STRING CAESAR_TOOL;
```

The global variable **CAESAR_TOOL** is a pointer to a character string which should contain the name of the *OPEN/CAESAR* application program. By default, **CAESAR_TOOL** is always initialized to the empty string **""**. Although this default value can be left unchanged, it is advisable for each application program to give a meaningful value to **CAESAR_TOOL**. This is generally done by the first instruction of the **main()** routine:

```
int main (argc, argv)
int  argc;
char *argv[];
{
/* additional declarations */
CAESAR_TOOL = argv[0];
/* other instructions */
}
```

....

CAESAR_WARNING

```
void CAESAR_WARNING (CAESAR_TYPE_STRING CAESAR_FORMAT, ...)
   { ... }
```

This function displays a (non-fatal) warning message to the standard output. The warning message is specified by the actual parameters passed to the function. These parameters follow the same conventions as for **printf(3)** and their number can be variable. The warning message will be prefixed by the value of the global variable **CAESAR_TOOL**, unless this value is the empty string. The format string given by **CAESAR_FORMAT** should not be suffixed with an end-of-line character, as **CAESAR_WARNING()** will add one automatically.

....

CAESAR_ERROR

```
void CAESAR_ERROR (CAESAR_TYPE_STRING CAESAR_FORMAT, ...)
   { ... }
```

This function displays a (fatal) error message to the standard output and stops using **exit (1)**. The error message is specified by the actual parameters passed to the function. These parameters follow the same conventions as for **printf(3)** and their number can be variable. The error message will be prefixed by the value of the global variable **CAESAR_TOOL**, unless this value is the empty string. The format string given by **CAESAR_FORMAT** should not be suffixed with an end-of-line character, as **CAESAR_ERROR()** will add one automatically.

....

CAESAR_PROTEST

```
#define CAESAR_PROTEST() ...
```

This macro-definition displays an error message (containing the current file name and current line number) and stops. This macro-definition should be used only to report about unexpected, internal errors. If a more detailed error message should be displayed, then function **CAESAR_ERROR()** should be used instead.

....

CAESAR_ASSERT

```
#define CAESAR_ASSERT(CAESAR_ASSERTION) \
        { if (!(CAESAR_ASSERTION)) CAESAR_PROTEST(); }
```

This macro-definition evaluates the boolean expression **CAESAR_ASSERTION** and, if the result is false, displays an error message (containing the current file name and current line number) and stops. This macro-definition should be used only to check assertions and report about unexpected, internal errors. If a more detailed error message should be displayed, then function **CAESAR_ERROR()** should be used instead.

....

CAESAR_TYPE_SIGNAL_HANDLER

```
typedef void (*CAESAR_TYPE_SIGNAL_HANDLER) (int);
```

CAESAR_TYPE_SIGNAL_HANDLER is the “pointer to a signal-handler procedure” type used in *OPEN/CAESAR*. A signal-handler procedure takes one parameter, which a POSIX signal number. The type **CAESAR_TYPE_SIGNAL_HANDLER** is identical to the type **sighandler_t** that exists in certain Unix distributions. See the **signal** manual page for further details.

....

CAESAR_SET_SIGNALS

```
void CAESAR_SET_SIGNALS (CAESAR_HANDLER)
   CAESAR_TYPE_SIGNAL_HANDLER CAESAR_HANDLER;
   { ... }
```

This procedure sets the POSIX signal handler to **CAESAR_HANDLER** for the following signals: **SIGHUP**, **SIGINT**, **SIGQUIT**, **SIGILL**, **SIGABRT**, **SIGFPE**, **SIGBUS**, **SIGSEGV**, **SIGSYS**, **SIGTERM**, and **SIGPIPE** (whenever these signals are supported by the operating system implementation). In particular, the signal handler **CAESAR_HANDLER** can be equal to the predefined values **SIG_IGN** or **SIG_DFL**. See the **signal** manual page for further details.

....

CAESAR_RESET_SIGNALS

```
void CAESAR_RESET_SIGNALS ()
   { ... }
```

This procedure resets the POSIX signal handler for the following signals: **SIGHUP**, **SIGINT**, **SIGQUIT**, **SIGILL**, **SIGABRT**, **SIGFPE**, **SIGBUS**, **SIGSEGV**, **SIGSYS**, **SIGTERM**, and **SIGPIPE** (whenever these signals are supported by the operating system implementation). If the procedure **CAESAR_SET_SIGNALS** has been invoked (at least once) and if no invocation of **CAESAR_RESET_SIGNALS** occurred after the most recent invocation of **CAESAR_SET_SIGNALS**, this signal handler is reset to the value of the **CAESAR_HANDLER** argument passed to the most recent invocation of **CAESAR_SET_SIGNALS**. If the procedure **CAESAR_SET_SIGNALS** has never been invoked or not been invoked after the most recent invocation of **CAESAR_RESET_SIGNALS**, this signal handler is reset to **SIG_DFL**.

....

CAESAR_TEMPORARY_FILE

```
CAESAR_TYPE_STRING CAESAR_TEMPORARY_FILE (CAESAR_TEMPORARY_SUFFIX)
   CAESAR_TYPE_STRING CAESAR_TEMPORARY_SUFFIX;
   { ... }
```

This function returns a character string that can safely be used as a file name for a temporary file. The prefix of this file name (i.e., the directory in which the temporary file will be created) is either given by the environment variable **$CADP_TMP** if this variable is defined, or is equal to **"/tmp"** otherwise. The suffix of this file name is given by **CAESAR_TEMPORARY_SUFFIX**, which is usually a file extension starting with a dot. If the allocation of the file name fails (due to a lack of memory) or if the temporary file cannot not be created in the directory specified by **$CADP_TMP** (e.g., because this directory does not exist, is write-protected, or belongs to a file system with unsufficient disk space to create new files), the **NULL** value is returned.

The contents of the character string returned by **CAESAR_TEMPORARY_FILE()** differs at every call to this function.

Note: It is not allowed to modify the character string returned by **CAESAR_TEMPORARY_FILE()** nor to free it, for instance using **free(3)**.

Note: The contents of the character string returned by **CAESAR_TEMPORARY_FILE()** may be destroyed by a subsequent call to this function.

....

CAESAR_OPEN_COMPRESSED_FILE

```
void CAESAR_OPEN_COMPRESSED_FILE (CAESAR_FILE, CAESAR_FILENAME, CAESAR_MODE)
   CAESAR_TYPE_FILE *CAESAR_FILE;
```

```
    CAESAR_TYPE_STRING CAESAR_FILENAME;
    CAESAR_TYPE_STRING CAESAR_MODE;
    { ... }
```

This function opens (for reading or writing) a file, the pathname of which is **CAESAR_FILENAME**. The value of **CAESAR_MODE**, which must be equal to **"r"** or **"w"**, determines whether this file is opened for reading (**"r"**) or writing (**"w"**).

If the suffix of **CAESAR_FILENAME** corresponds to a known compression format (**".Z"**, **".gz"**, **".bz2"**, etc.), the file will be treated as a compressed file. Otherwise, the file will not be compressed and treated as an ordinary file. The list of supported suffixes and compression formats is given by the **cadp_zip** shell-script provided within the CADP distribution.

If the file can be opened, the value of ∗**CAESAR_FILE** is set to a POSIX stream descriptor from (respectively, to) which data can be read (respectively, written) using the standard input (respectively, output) routines available in POSIX, e.g., **fgets(3)**, **fprintf(3)**, **fputs(3)**, **fscanf(3)**, etc. Data compression, if any, is performed transparently, meaning that the data sent or read to a compressed file are exactly the same as if the file was not compressed.

If the file cannot be opened, the value of ∗**CAESAR_FILE** is set to **NULL**.

Note: Since type **CAESAR_TYPE_FILE** is defined as **FILE** ∗, variable **CAESAR_FILE** is actually of type **FILE** ∗∗, where **FILE** is the type of a POSIX file descriptor.

Note: Function **CAESAR_OPEN_COMPRESSED_FILE()** invokes **fopen(3)** to open an uncompressed file and **popen(3)** to open a compressed file.

....

CAESAR_CLOSE_COMPRESSED_FILE

```
void CAESAR_CLOSE_COMPRESSED_FILE (CAESAR_FILE)
    CAESAR_TYPE_FILE *CAESAR_FILE;
    { ... }
```

This function closes the file pointed to by ∗**CAESAR_FILE**, which must have been opened previously by a call to the **CAESAR_OPEN_COMPRESSED_FILE()** function. After the file is closed, the value of ∗**CAESAR_FILE** is set to **NULL**.

Note: Since type **CAESAR_TYPE_FILE** is defined as **FILE** ∗, variable **CAESAR_FILE** is actually of type **FILE** ∗∗, where **FILE** is the type of a POSIX file descriptor.

Note: Function **CAESAR_CLOSE_COMPRESSED_FILE()** invokes either **fclose(3)** or **pclose(3)** to close the file.

....

CAESAR_FUNCTION_NAME

```
CAESAR_TYPE_STRING CAESAR_FUNCTION_NAME (CAESAR_FUNCTION)
    void (*CAESAR_FUNCTION) ();
    { ... }
```

This function returns a printable character string identifying the function **CAESAR_FUNCTION**. If

`CAESAR_FUNCTION` belongs to a predefined set of functions of the *OPEN/CAESAR* library, the character string contains the name of this function (e.g., **`CAESAR_0_HASH`**, **`CAESAR_PRINT_STATE`**, **`CAESAR_COMPARE_LABEL`**, etc.); otherwise, the character string contains the hexadecimal value of the function pointer **`CAESAR_FUNCTION`**.

Note: It is not allowed to modify the character string returned by **`CAESAR_FUNCTION_NAME()`** nor to free it, for instance using **`free(3)`**.

Note: In fact, **`CAESAR_FUNCTION_NAME()`** is implemented as a macro-definition that invokes an auxiliary function. This avoids the need for inserting a type cast before any parameter given to **`CAESAR_FUNCTION_NAME()`**.

....

AUTHOR(S)

Hubert Garavel

FILES

$CADP/incl/caesar_graph.h	interface of the graph module
$CADP/incl/caesar_*.h	interfaces of the storage module
$CADP/bin.‘arch‘/libcaesar.a	object code of the storage module
$CADP/src/open_caesar/*.c	source code of various exploration modules
$CADP/com/lotos.open	shell script to run OPEN/CAESAR

SEE ALSO

Reference Manuals of OPEN/CAESAR, CAESAR, and CAESAR.ADT, **lotos.open**(LOCAL), **caesar**(LOCAL), **caesar.adt**(LOCAL)

Additional information is available from the CADP Web page located at http://cadp.inria.fr

Directives for installation are given in files **$CADP/INSTALLATION_*.**

Recent changes and improvements to this software are reported and commented in file **$CADP/HISTORY.**

BUGS

Known bugs are described in the Reference Manual of OPEN/CAESAR. Please report new bugs to cadp@inria.fr