NAME

caesar_area_1 – the “area_1” library of OPEN/CAESAR

PURPOSE

The “area_1” library provides primitives for managing “areas”, which are memory chunks of various sizes and alignment factors. This library provides for genericity by allowing different objects (states, labels, character strings, user-defined memory chunks) to be handled uniformly.

USAGE

The “area_1” library consists of:

- a predefined header file **`caesar_area_1.h`**;
- the precompiled library file **`libcaesar.a`**, which implements the features described in **`caesar_area_1.h`**.

Note: The “area_1” library is a software layer built above the primitives offered by the “standard” and “hash” libraries, and by the *OPEN/CAESAR* graph module.

DESCRIPTION

An “area” is basically a memory chunk (i.e., a sequence of contiguous bytes) characterized by its (fixed) size and its alignment factor (see the description of **`CAESAR_ALIGNMENT_POINTER()`** in the “standard” library for a definition of the alignment factor).

There are five different kinds of areas:

- an “ordinary area” may contain any kind of data; the precise contents of an ordinary area is left to the user and is not specified in the context of the “area_1” library;
- an “empty area” is a special area of zero bytes;
- a “state area” is a special area dedicated to contain a state, as described in the graph module, i.e., a value of type **`CAESAR_BODY_STATE`**;
- a “label area” is a special area dedicated to contain a label, as described in the graph module, i.e., a value of type **`CAESAR_BODY_LABEL`**;
- a “string area” is a special area dedicated to contain a pointer to a null-terminated character string, i.e., a value of type **`CAESAR_TYPE_STRING`**. Notice that a string area does not contain a (variable length) character string but a (fixed length) pointer to a (variable length) character string.

FEATURES

....

`CAESAR_TYPE_AREA_1`

```
typedef CAESAR_TYPE_NATURAL CAESAR_TYPE_AREA_1;
#define CAESAR_EXPONENT_EMPTY_AREA_1      0
#define CAESAR_EXPONENT_STATE_AREA_1      1
#define CAESAR_EXPONENT_LABEL_AREA_1      2
#define CAESAR_EXPONENT_STRING_AREA_1     3
```

Concretely, the type **`CAESAR_TYPE_AREA_1`** is represented as a numerical type (32-bit or 64-bit natural number depending on the machine architecture). Logically, a value of type **`CAESAR_TYPE_AREA_1`** is a pair (length field, exponent field), where:

- the exponent field is an unsigned natural number coded on the 4 highest bits (thus, in the range

0..15), and

- the length field is an unsigned natural number coded on the remaining (all but 4) lowest bits, i.e., either coded on 28 bits (and thus, in the range 0...268,435,455) on 32-bit machines, or coded on 60 bits (and thus, in the range 0...1,152,921,504,606,846,975) on 64-bit machines.

The different kinds of areas are represented as follows:

- for an ordinary area, the length field is different from zero and represents the size (in bytes) of the area; if the exponent field is equal to a value f different from zero, the alignment factor (in bytes) of the area is equal to 2^{f-1}; the case in which the exponent field is equal to zero corresponds to a backward compatibility situation, which is now obsolete and in which the alignment factor is guessed empirically from the value of the length field (see the definition of **CAESAR_ALIGNMENT_AREA_1()** below);
- for an empty area, the length field is equal to zero and the exponent field is equal to the constant **CAESAR_EXPONENT_EMPTY_AREA_1**;
- for a state area, the length field is equal to zero and the exponent field is equal to the constant **CAESAR_EXPONENT_STATE_AREA_1**;
- for a label area, the length field is equal to zero and the exponent field is equal to the constant **CAESAR_EXPONENT_LABEL_AREA_1**;
- for a string area, the length field is equal to zero and the exponent field is equal to the constant **CAESAR_EXPONENT_STRING_AREA_1**;
- any other case in which the length field is equal to zero is undefined.

....

CAESAR_LENGTH_AREA_1

```
CAESAR_TYPE_NATURAL CAESAR_LENGTH_AREA_1 (CAESAR_AREA)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   { ... }
```

This function returns the length field of the area **CAESAR_AREA**.

....

CAESAR_EXPONENT_AREA_1

```
CAESAR_TYPE_NATURAL CAESAR_EXPONENT_AREA_1 (CAESAR_AREA)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   { ... }
```

This function returns the exponent field of the area **CAESAR_AREA**.

....

CAESAR_AREA_1

```
CAESAR_AREA_1 CAESAR_AREA_1 (CAESAR_LENGTH, CAESAR_EXPONENT)
   CAESAR_TYPE_NATURAL CAESAR_LENGTH;
   CAESAR_TYPE_NATURAL CAESAR_EXPONENT;
   { ... }
```

This function returns the area with a length field equal to **CAESAR_LENGTH** and an exponent field equal to

CAESAR_EXPONENT.

Note: this function does not check that its parameters are consistent, e.g., that for an ordinary area the alignment (specified by the exponent field) is an exact divider of the size (specified by the length field).

....

CAESAR_EMPTY_AREA_1

```
CAESAR_TYPE_AREA_1 CAESAR_EMPTY_AREA_1 ()
   { ... }
```

This function returns the area with a length field equal to 0 and an exponent field equal to **CAESAR_EXPONENT_EMPTY_AREA_1** (i.e., corresponding to an empty area).

Note: For backward compatibility reasons, **CAESAR_EMPTY_AREA_1()** is equal to 0.

....

CAESAR_STATE_AREA_1

```
CAESAR_TYPE_AREA_1 CAESAR_STATE_AREA_1 ()
   { ... }
```

This function returns the area with a length field equal to 0 and an exponent field equal to **CAESAR_EXPONENT_STATE_AREA_1** (i.e., corresponding to a state area).

....

CAESAR_LABEL_AREA_1

```
CAESAR_TYPE_AREA_1 CAESAR_LABEL_AREA_1 ()
   { ... }
```

This function returns the area with a length field equal to 0 and an exponent field equal to **CAESAR_EXPONENT_LABEL_AREA_1** (i.e., corresponding to a label area).

....

CAESAR_STRING_AREA_1

```
CAESAR_TYPE_AREA_1 CAESAR_STRING_AREA_1 ()
   { ... }
```

This function returns the area with a length field equal to 0 and an exponent field equal to **CAESAR_EXPONENT_STRING_AREA_1** (i.e., corresponding to a string area).

....

CAESAR_BYTE_AREA_1

```
CAESAR_TYPE_AREA_1 CAESAR_BYTE_AREA_1 (CAESAR_LENGTH)
   CAESAR_TYPE_NATURAL CAESAR_LENGTH;
   { ... }
```

This function returns the area with a length field equal to **CAESAR_LENGTH** and an exponent field corresponding to a byte or a character (i.e., a memory area with an alignment of 1).

....

CAESAR_NATURAL_AREA_1

```
CAESAR_TYPE_AREA_1 CAESAR_NATURAL_AREA_1 (CAESAR_LENGTH)
   CAESAR_TYPE_NATURAL CAESAR_LENGTH;
   { ... }
```

This function returns the area with a length field equal to **CAESAR_LENGTH** and an exponent field corresponding to a natural or integer number (i.e., a memory area with an alignment suitable for a value of type **CAESAR_TYPE_NATURAL** or **CAESAR_TYPE_INTEGER**).

Note: In principle, such an area should not contain pointers.

....

CAESAR_POINTER_AREA_1

```
CAESAR_TYPE_AREA_1 CAESAR_POINTER_AREA_1 (CAESAR_LENGTH)
   CAESAR_TYPE_NATURAL CAESAR_LENGTH;
   { ... }
```

This function returns the area with a length field equal to **CAESAR_LENGTH** and an exponent field corresponding to a pointer (i.e., a memory area with an alignment equal to **CAESAR_ALIGNMENT_POINTER()**).

Note: In principle, such an area should contain at least one pointer, because pointers usually have the strongest alignment constraints.

....

CAESAR_SIZE_AREA_1

```
CAESAR_TYPE_NATURAL CAESAR_SIZE_AREA_1 (CAESAR_AREA)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   { ... }
```

This function returns the size (in bytes) of the area **CAESAR_AREA**:

- for an ordinary area, this size is equal to the length field of **CAESAR_AREA**;
- for an empty area, this size is equal to zero;
- for a state area, this size is given by the **CAESAR_SIZE_STATE()** function exported by the graph module;
- for a label area, this size is given by the **CAESAR_SIZE_LABEL()** function exported by the graph module;
- for a string area, this size is equal to **CAESAR_SIZE_POINTER()** (which corresponds to the size of a character string pointer).

....

CAESAR_HASH_SIZE_AREA_1

```
CAESAR_TYPE_NATURAL CAESAR_HASH_SIZE_AREA_1 (CAESAR_AREA)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   { ... }
```

This function returns the "hashable" size (in bytes) of the area **CAESAR_AREA**:

- for an ordinary area, this size is equal to the length field of **CAESAR_AREA** (this is an arbitrary definition, since the actual contents of **CAESAR_AREA** are unknown);
- for an empty area, this size is equal to zero (which expresses the fact that an empty area is not appropriate for hashing);
- for a state area, this size is given by the **CAESAR_HASH_SIZE_STATE()** function exported by the graph module;
- for a label area, this size is given by the **CAESAR_HASH_SIZE_LABEL()** function exported by the graph module;
- for a string area, this size is equal to zero (which expresses the fact that a character string pointer, i.e., a value of type **(CAESAR_TYPE_STRING *)**, is not directly appropriate for hashing).

....

CAESAR_ALIGNMENT_AREA_1

```
CAESAR_TYPE_NATURAL CAESAR_ALIGNMENT_AREA_1 (CAESAR_AREA)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   { ... }
```

This function returns the alignment factor (in bytes) of the area **CAESAR_AREA**:

- for an ordinary area whose exponent field f is different from zero, the alignment factor is equal to 2^{f-1};
- for an ordinary area whose exponent field is equal to zero (backward compatibility case), the alignment factor is guessed empirically from the size of **CAESAR_AREA** (precisely, the alignment factor will be the largest value in the set {1, 2, 4, 8} that divides the size of **CAESAR_AREA** exactly; this is an arbitrary definition, since the actual contents of **CAESAR_AREA** are unknown);
- for an empty area, this alignment factor is equal to one (which expresses the fact that an empty area has no specific alignment constraint);
- for a state area, this alignment factor is given by the **CAESAR_ALIGNMENT_STATE()** function exported by the graph module;
- for a label area, this alignment factor is given by the **CAESAR_ALIGNMENT_LABEL()** function exported by the graph module;
- for a string area, this alignment factor is equal to **CAESAR_ALIGNMENT_POINTER()** (which corresponds to the alignment factor of a character string pointer).

....

CAESAR_COPY_AREA_1

```
void CAESAR_COPY_AREA_1 (CAESAR_P1, CAESAR_P2, CAESAR_SIZE)
   CAESAR_TYPE_POINTER CAESAR_P1;
   CAESAR_TYPE_POINTER CAESAR_P2;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   { ... }
```

This procedure copies the memory area (of **CAESAR_SIZE** bytes) pointed to by **CAESAR_P2** to the location pointed to by **CAESAR_P1**.

Note: This function is implemented as a simple wrapper for the POSIX function **memcpy(3)**.

....

CAESAR_COMPARE_EMPTY_AREA_1

```
CAESAR_TYPE_BOOLEAN CAESAR_COMPARE_EMPTY_AREA_1 (CAESAR_P1, CAESAR_P2)
   CAESAR_TYPE_POINTER CAESAR_P1;
   CAESAR_TYPE_POINTER CAESAR_P2;
   { ... }
```

This function is intended to compare two empty areas and, thus, always returns a value different from 0 (since there is only one empty area).

Note: this function is used by the **CAESAR_USE_COMPARE_FUNCTION_AREA_1()** function described below.

....

CAESAR_COMPARE_STRING_AREA_1

```
CAESAR_TYPE_BOOLEAN CAESAR_COMPARE_STRING_AREA_1 (CAESAR_S1, CAESAR_S2)
   CAESAR_TYPE_STRING *CAESAR_S1;
   CAESAR_TYPE_STRING *CAESAR_S2;
   { ... }
```

This function returns a value different from 0 if both character strings pointed to by ***CAESAR_S1** and ***CAESAR_S2** are identical, or 0 if they are not.

Note: This function uses the POSIX function **strcmp(3)**.

Note: this function is used by the **CAESAR_USE_COMPARE_FUNCTION_AREA_1()** function described below.

....

CAESAR_USE_COMPARE_FUNCTION_AREA_1

```
void CAESAR_USE_COMPARE_FUNCTION_AREA_1 (CAESAR_AREA, CAESAR_COMPARE_FUNCTION)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   CAESAR_TYPE_COMPARE_FUNCTION *CAESAR_COMPARE_FUNCTION;
   { ... }
```

This procedure modifies the (function pointer) value pointed to by ***CAESAR_COMPARE_FUNCTION** (possibly to assign this function pointer a default value if it is equal to **NULL** before the procedure is invoked) in the following cases:

- if **CAESAR_AREA** is an empty area and if ***CAESAR_COMPARE_FUNCTION** is equal to **NULL**, then the **CAESAR_COMPARE_EMPTY_AREA_1()** function defined above will be assigned to ***CAESAR_COMPARE_FUNCTION**;

Note: it is not allowed to call this procedure if **CAESAR_AREA** is an empty area, and if ***CAESAR_COMPARE_FUNCTION** is different from both **NULL** and **CAESAR_COMPARE_EMPTY_AREA_1 ()**, since in this case **CAESAR_COMPARE_EMPTY_AREA_1 ()** is the only sensible comparison function;

- if **CAESAR_AREA** is a state area and if ***CAESAR_COMPARE_FUNCTION** is equal to **NULL**, then the **CAESAR_COMPARE_STATE ()** function of the graph module will be assigned to ***CAESAR_COMPARE_FUNCTION**;

  Note: it is not allowed to call this procedure if **CAESAR_AREA** is a state area, and if ***CAESAR_COMPARE_FUNCTION** is different from both **NULL** and **CAESAR_COMPARE_STATE ()**, and if the result of **CAESAR_HASH_SIZE_STATE ()** is strictly less than the result of **CAESAR_SIZE_STATE ()**, since in this case **CAESAR_COMPARE_STATE ()** is the only sensible comparison function;
- if **CAESAR_AREA** is a label area and if ***CAESAR_COMPARE_FUNCTION** is equal to **NULL**, then the **CAESAR_COMPARE_LABEL ()** function of the graph module will be assigned to ***CAESAR_COMPARE_FUNCTION**;

  Note: it is not allowed to call this procedure if **CAESAR_AREA** is a label area, and if ***CAESAR_COMPARE_FUNCTION** is different from both **NULL** and **CAESAR_COMPARE_LABEL ()**, and if the result of **CAESAR_HASH_SIZE_LABEL ()** is strictly less than the result of **CAESAR_SIZE_LABEL ()**, since in this case **CAESAR_COMPARE_LABEL ()** is the only sensible comparison function;
- if **CAESAR_AREA** is a string area and if ***CAESAR_COMPARE_FUNCTION** is equal to **NULL**, then the **CAESAR_COMPARE_STRING_AREA_1 ()** function defined above will be assigned to ***CAESAR_COMPARE_FUNCTION**.

In any other case, ***CAESAR_COMPARE_FUNCTION** is kept unchanged.

....

CAESAR_COMPARE_AREA_1

```
CAESAR_TYPE_BOOLEAN CAESAR_COMPARE_AREA_1 (CAESAR_COMPARE_FUNCTION,
                                           CAESAR_P1, CAESAR_P2, CAESAR_SIZE)
   CAESAR_TYPE_COMPARE_FUNCTION CAESAR_COMPARE_FUNCTION;
   CAESAR_TYPE_POINTER CAESAR_P1;
   CAESAR_TYPE_POINTER CAESAR_P2;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   { ... }
```

This function compares the two memory areas (of **CAESAR_SIZE** bytes) pointed to by **CAESAR_P1** and **CAESAR_P2** using either **CAESAR_COMPARE_FUNCTION** if this function pointer is not **NULL**, or the POSIX function **memcmp(3)** otherwise. The result is **CAESAR_TRUE** if and only if both memory areas are found to be equal. Note: Before calling **CAESAR_COMPARE_AREA_1 ()**, the actual value of **CAESAR_COMPARE_FUNCTION** should have been set by calling the **CAESAR_USE_COMPARE_FUNCTION_AREA_1 ()** function.

....

CAESAR_HASH_EMPTY_AREA_1

```
CAESAR_TYPE_BOOLEAN CAESAR_HASH_EMPTY_AREA_1 (CAESAR_P, CAESAR_MODULUS)
```

```
    CAESAR_TYPE_POINTER CAESAR_P;
    CAESAR_TYPE_NATURAL CAESAR_MODULUS;
    { ... }
```

This function is intended to compute an hash-value on empty areas and, thus, always returns 0 (since there is only one empty area). If **CAESAR_MODULUS** is equal to 0, the result is undefined.

Note: this function is used by the **CAESAR_USE_HASH_FUNCTION_AREA_1()** function described below.

....

CAESAR_HASH_STRING_AREA_1

```
CAESAR_TYPE_BOOLEAN CAESAR_HASH_STRING_AREA_1 (CAESAR_P, CAESAR_MODULUS)
    CAESAR_TYPE_POINTER CAESAR_P;
    CAESAR_TYPE_NATURAL CAESAR_MODULUS;
    { ... }
```

This function is intended to compute an hash-value on string areas. It returns a value in the range 0..(**CAESAR_MODULUS**-1), computed from the character string pointed to by ***CAESAR_P** (and not by **CAESAR_P**, as **CAESAR_P** is expected to be of type **(CAESAR_TYPE_STRING *)**, not **CAESAR_TYPE_STRING**).

Note: to compute the hash-value, this function invokes the **CAESAR_STRING_0_HASH()** function of the ‘‘hash’’ library.

Note: this function is used by the **CAESAR_USE_HASH_FUNCTION_AREA_1()** function described below.

....

CAESAR_USE_HASH_FUNCTION_AREA_1

```
void CAESAR_USE_HASH_FUNCTION_AREA_1 (CAESAR_AREA, CAESAR_HASH_FUNCTION)
    CAESAR_TYPE_AREA_1 CAESAR_AREA;
    CAESAR_TYPE_HASH_FUNCTION *CAESAR_HASH_FUNCTION;
    { ... }
```

This procedure modifies the (function pointer) value pointed to by ***CAESAR_HASH_FUNCTION** (possibly to assign this function pointer a default value if it is equal to **NULL** before the procedure is invoked) in the following cases:

- if **CAESAR_AREA** is an empty area and if ***CAESAR_HASH_FUNCTION** is equal to **NULL**, then the **CAESAR_HASH_EMPTY_AREA_1()** function defined above will be assigned to ***CAESAR_HASH_FUNCTION**;

  Note: it is not allowed to call this procedure if **CAESAR_AREA** is an empty area, and if ***CAESAR_HASH_FUNCTION** is different from both **NULL** and **CAESAR_HASH_EMPTY_AREA_1()**, since in this case **CAESAR_HASH_EMPTY_AREA_1()** is the only sensible hashing function;
- if **CAESAR_AREA** is a state area and if ***CAESAR_HASH_FUNCTION** is equal to **NULL**, then the **CAESAR_HASH_STATE()** function of the graph module will be assigned to ***CAESAR_HASH_FUNCTION**;

Note: it is not allowed to call this procedure if **CAESAR_AREA** is a state area, and if ***CAESAR_HASH_FUNCTION** is different from both **NULL** and **CAESAR_HASH_STATE()**, and if the result of **CAESAR_HASH_SIZE_STATE()** is equal to zero, since in this case **CAESAR_HASH_STATE()** is the only sensible hashing function;

- if **CAESAR_AREA** is a label area and if ***CAESAR_HASH_FUNCTION** is equal to **NULL**, then the **CAESAR_HASH_LABEL()** function of the graph module will be assigned to ***CAESAR_HASH_FUNCTION**;

  Note: it is not allowed to call this procedure if **CAESAR_AREA** is a label area, and if ***CAESAR_HASH_FUNCTION** is different from both **NULL** and **CAESAR_HASH_LABEL()**, and if the result of **CAESAR_HASH_SIZE_LABEL()** is equal to zero, since in this case **CAESAR_HASH_LABEL()** is the only sensible hashing function;
- if **CAESAR_AREA** is a string area and if ***CAESAR_HASH_FUNCTION** is equal to **NULL**, then the **CAESAR_HASH_STRING_AREA_1()** function of the ‘‘hash’’ library will be assigned to ***CAESAR_HASH_FUNCTION**.

In any other case, ***CAESAR_HASH_FUNCTION** is kept unchanged.

....

CAESAR_HASH_AREA_1

```
CAESAR_TYPE_NATURAL CAESAR_HASH_AREA_1 (CAESAR_HASH_FUNCTION, CAESAR_P,
                                        CAESAR_HASH_SIZE, CAESAR_MODULUS)
   CAESAR_TYPE_HASH_FUNCTION CAESAR_HASH_FUNCTION;
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_HASH_SIZE;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash value for the memory area pointed to by **CAESAR_P** using either **CAESAR_HASH_FUNCTION** if this function pointer is not **NULL**, or function **CAESAR_0_HASH()** of the ‘‘hash’’ library otherwise. Hashing is performed on the **CAESAR_HASH_SIZE** first bytes and the hash value returned is in the range 0..(**CAESAR_MODULUS**-1).

Note: Before calling **CAESAR_HASH_AREA_1()**, the actual value of **CAESAR_HASH_FUNCTION** should have been set by calling the **CAESAR_USE_HASH_FUNCTION_AREA_1()** function.

....

CAESAR_CONVERT_EMPTY_AREA_1

```
CAESAR_TYPE_STRING CAESAR_CONVERT_EMPTY_AREA_1 (CAESAR_P)
   CAESAR_TYPE_POINTER CAESAR_P;
   { ... }
```

This function returns a constant, user-readable character string representing the empty area.

Note: It is not allowed to modify the character string returned by **CAESAR_CONVERT_EMPTY_AREA_1()** nor to free it, for instance using **free(3)**.

Note: this function is used by the **CAESAR_USE_CONVERT_FUNCTION_AREA_1()** function described below.

....

CAESAR_CONVERT_STRING_AREA_1

```
CAESAR_TYPE_STRING CAESAR_CONVERT_STRING_AREA_1 (CAESAR_S)
   CAESAR_TYPE_STRING *CAESAR_S;
   { ... }
```

This function returns the character string pointed to by ***CAESAR_S**.

Note: this function is used by the **CAESAR_USE_CONVERT_FUNCTION_AREA_1()** function described below.

....

CAESAR_CONVERT_BINARY_AREA_1

```
CAESAR_TYPE_STRING CAESAR_CONVERT_BINARY_AREA_1 (CAESAR_P, CAESAR_SIZE)
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   { ... }
```

This function returns a pointer to a character string corresponding to the hexadecimal representation for the **CAESAR_SIZE** bytes-long memory chunk pointed to by **CAESAR_P**.

Note: It is not allowed to modify the character string returned by **CAESAR_CONVERT_BINARY_AREA_1()** nor to free it, for instance using **free(3)**.

Note: this function is used by the **CAESAR_CONVERT_AREA_1()** function described below.

....

CAESAR_USE_CONVERT_FUNCTION_AREA_1

```
void CAESAR_USE_CONVERT_FUNCTION_AREA_1 (CAESAR_AREA, CAESAR_CONVERT_FUNCTION)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   CAESAR_TYPE_CONVERT_FUNCTION *CAESAR_CONVERT_FUNCTION;
   { ... }
```

This procedure modifies the (function pointer) value pointed to by ***CAESAR_CONVERT_FUNCTION** (possibly to assign this function pointer a default value if it is equal to **NULL** before the procedure is invoked) in the following cases:

- if **CAESAR_AREA** is an empty area and if ***CAESAR_CONVERT_FUNCTION** is equal to **NULL**, then the **CAESAR_CONVERT_EMPTY_AREA_1()** function defined above will be assigned to ***CAESAR_CONVERT_FUNCTION**;
- if **CAESAR_AREA** is a state area and if ***CAESAR_CONVERT_FUNCTION** is equal to **NULL**, then the effect of this procedure is undefined;
- if **CAESAR_AREA** is a label area and if ***CAESAR_CONVERT_FUNCTION** is equal to **NULL**, then the **CAESAR_STRING_LABEL()** function of the graph module will be assigned to ***CAESAR_CONVERT_FUNCTION**;
- if **CAESAR_AREA** is a string area and if ***CAESAR_CONVERT_FUNCTION** is equal to **NULL**, then the **CAESAR_CONVERT_STRING_AREA_1()** function defined above will be assigned to

***CAESAR_CONVERT_FUNCTION**.

In any other case, ***CAESAR_CONVERT_FUNCTION** is kept unchanged.

....

CAESAR_CONVERT_AREA_1

```
CAESAR_TYPE_STRING CAESAR_CONVERT_AREA_1 (CAESAR_CONVERT_FUNCTION, CAESAR_P,
                                          CAESAR_SIZE)
   CAESAR_TYPE_CONVERT_FUNCTION CAESAR_CONVERT_FUNCTION;
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   { ... }
```

This function converts the memory area (of **CAESAR_SIZE** bytes) pointed to by **CAESAR_P** into a character string using either **CAESAR_CONVERT_FUNCTION** if this function pointer is not **NULL**, or function **CAESAR_CONVERT_BINARY_AREA_1()** otherwise.

Note: Before calling **CAESAR_CONVERT_AREA_1()**, the actual value of **CAESAR_CONVERT_FUNCTION** should have been set by calling the **CAESAR_USE_CONVERT_FUNCTION_AREA_1()** function.

....

CAESAR_PRINT_EMPTY_AREA_1

```
void CAESAR_PRINT_EMPTY_AREA_1 (CAESAR_FILE, CAESAR_P)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_POINTER CAESAR_P;
   { ... }
```

This procedure prints to file **CAESAR_FILE** a constant, user-readable character string representing the empty area.

Before this procedure is called, **CAESAR_FILE** must have been properly opened, for instance using **fopen(3)**.

Note: this function is used by the **CAESAR_USE_PRINT_FUNCTION_AREA_1()** function described below.

....

CAESAR_PRINT_STRING_AREA_1

```
void CAESAR_PRINT_STRING_AREA_1 (CAESAR_FILE, CAESAR_S)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_STRING *CAESAR_S;
   { ... }
```

This procedure prints to file **CAESAR_FILE** the character string pointed to by ***CAESAR_S**.

Before this procedure is called, **CAESAR_FILE** must have been properly opened, for instance using **fopen(3)**.

Note: this function is used by the **CAESAR_USE_PRINT_FUNCTION_AREA_1()** function described below.

....

CAESAR_PRINT_BINARY_AREA_1

```
void CAESAR_PRINT_BINARY_AREA_1 (CAESAR_FILE, CAESAR_P, CAESAR_SIZE)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   { ... }
```

This procedure prints to file **CAESAR_FILE** the character string corresponding to the hexadecimal representation for the **CAESAR_SIZE** bytes-long memory chunk pointed to by **CAESAR_P**.

Before this procedure is called, **CAESAR_FILE** must have been properly opened, for instance using **fopen(3)**.

Note: this function is used by the **CAESAR_PRINT_AREA_1()** function described below.

....

CAESAR_USE_PRINT_FUNCTION_AREA_1

```
void CAESAR_USE_PRINT_FUNCTION_AREA_1 (CAESAR_AREA, CAESAR_PRINT_FUNCTION)
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   CAESAR_TYPE_PRINT_FUNCTION *CAESAR_PRINT_FUNCTION;
   { ... }
```

This procedure modifies the (function pointer) value pointed to by ***CAESAR_PRINT_FUNCTION** (possibly to assign this function pointer a default value if it is equal to **NULL** before the procedure is invoked) in the following cases:

- if **CAESAR_AREA** is an empty area and if ***CAESAR_PRINT_FUNCTION** is equal to **NULL**, then the **CAESAR_PRINT_EMPTY_AREA_1()** function defined above will be assigned to ***CAESAR_PRINT_FUNCTION**;
- if **CAESAR_AREA** is a state area and if ***CAESAR_PRINT_FUNCTION** is equal to **NULL**, then the **CAESAR_PRINT_STATE()** function of the graph module will be assigned to ***CAESAR_PRINT_FUNCTION**;
- if **CAESAR_AREA** is a label area and if ***CAESAR_PRINT_FUNCTION** is equal to **NULL**, then the **CAESAR_PRINT_LABEL()** function of the graph module will be assigned to ***CAESAR_PRINT_FUNCTION**;
- if **CAESAR_AREA** is a string area and if ***CAESAR_PRINT_FUNCTION** is equal to **NULL**, then the **CAESAR_PRINT_STRING_AREA_1()** function defined above will be assigned to ***CAESAR_PRINT_FUNCTION**.

In any other case, ***CAESAR_PRINT_FUNCTION** is kept unchanged.

....

CAESAR_PRINT_AREA_1

```
void CAESAR_PRINT_AREA_1 (CAESAR_PRINT_FUNCTION, CAESAR_FILE, CAESAR_P,
                                             CAESAR_SIZE)
         CAESAR_TYPE_PRINT_FUNCTION CAESAR_PRINT_FUNCTION;
         CAESAR_TYPE_FILE CAESAR_FILE;
         CAESAR_TYPE_POINTER CAESAR_P;
         CAESAR_TYPE_NATURAL CAESAR_SIZE;
         { ... }
```

This function prints to file **CAESAR_FILE** a character string representing the memory area (of **CAESAR_SIZE** bytes) pointed to by **CAESAR_P**. Printing is done using either **CAESAR_PRINT_FUNCTION** if this function pointer is not **NULL**, or function **CAESAR_PRINT_BINARY_AREA_1()** otherwise.

Before this procedure is called, **CAESAR_FILE** must have been properly opened, for instance using **fopen(3)**.

Note: Before calling **CAESAR_PRINT_AREA_1()**, the actual value of **CAESAR_PRINT_FUNCTION** should have been set by calling the **CAESAR_USE_PRINT_FUNCTION_AREA_1()** function.

....

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