NAME

caesar_mask_1 – the ‘‘mask_1’’ library of OPEN/CAESAR

PURPOSE

The ‘‘mask_1’’ library provides primitives for applying sequences of hiding and renaming operations to memory areas (such as labels) converted to character strings. In particular, this allows to hide and/or rename labels on the fly.

USAGE

The ‘‘mask_1’’ library consists of:

- a predefined header file **`caesar_mask_1.h`**;
- the precompiled library file **`libcaesar.a`**, which implements the features described in **`caesar_mask_1.h`**.

Note: The ‘‘mask_1’’ library is a software layer built above the primitives offered by the ‘‘standard’’, ‘‘area_1’’, ‘‘hide_1’’, and ‘‘rename_1’’ libraries.

DESCRIPTION

The ‘‘mask_1’’ library allows to apply successive operations to memory areas (for instance, labels). These operations are, first, the conversion of a memory area to a character string, and then a sequence of hiding and/or renaming operations on the resulting character string.

Hiding a character string consists in renaming it into the internal action ‘‘i’’ (or tau) according to a ‘‘hiding file’’, which contains a list of regular expressions. The format of ‘‘hiding files’’ is defined in the ‘‘hide_1’’ library.

Renaming consists in applying string substitutions according to a ‘‘renaming file’’, which contains a list of substitution patterns defined using regular expressions. The format of ‘‘renaming files’’ is defined in the ‘‘rename_1’’ library.

The ‘‘mask_1’’ library targets at efficiency. Each hiding file and each renaming file is parsed and checked only once. In addition, a hash-based cache table is available to speed up performance; this table stores the character strings resulting from prior operations performed by the masking object, so as to avoid multiple, redundant computations.

The ‘‘mask_1’’ library introduces the concept of ‘‘masking object’’, which is an abstract data structure containing (among others) the following information:

- a description of the memory areas to be processed, including their size, hash size, comparison function, hashing function, conversion function, and printing procedure;
- a (possibly empty) ordered list of hiding and renaming operations to be applied sequentially; hiding and renaming operations can be intertwined arbitrarily;
- an (optional) hash-based cache table and (if applicable) statistical data about caching efficiency.

FEATURES

.....

`CAESAR_TYPE_MASK_1`

```
typedef CAESAR_TYPE_ABSTRACT (...) CAESAR_TYPE_MASK_1;
```

This type denotes a pointer to the concrete representation of a masking object, which is supposed to be

"opaque".

.....

CAESAR_CREATE_MASK_1

```
void CAESAR_CREATE_MASK_1 (CAESAR_M, CAESAR_AREA, CAESAR_CACHE,
                           CAESAR_CACHE_SIZE, CAESAR_PRIME, CAESAR_COMPARE,
                           CAESAR_HASH, CAESAR_CONVERT, CAESAR_HISTORY)
   CAESAR_TYPE_MASK_1 *CAESAR_M;
   CAESAR_TYPE_AREA_1 CAESAR_AREA;
   CAESAR_TYPE_BOOLEAN CAESAR_CACHE;
   CAESAR_TYPE_NATURAL CAESAR_CACHE_SIZE;
   CAESAR_TYPE_BOOLEAN CAESAR_PRIME;
   CAESAR_TYPE_COMPARE_FUNCTION CAESAR_COMPARE;
   CAESAR_TYPE_HASH_FUNCTION CAESAR_HASH;
   CAESAR_TYPE_CONVERT_FUNCTION CAESAR_CONVERT;
   CAESAR_TYPE_BOOLEAN CAESAR_HISTORY;
   { ... }
```

This procedure allocates a masking object using **CAESAR_CREATE()** and assigns its address to ***CAESAR_M**. If the allocation fails or if any error occurs in this procedure, the **NULL** value is assigned to ***CAESAR_M**.

Note: because **CAESAR_TYPE_MASK_1** is a pointer type, any variable **CAESAR_M** of type **CAESAR_TYPE_MASK_1** must be allocated before used, for instance using:

```
CAESAR_CREATE_MASK_1 (&CAESAR_M, ...);
```

The value of **CAESAR_AREA** specifies the kind of memory areas that can be processed by the masking object to be created. All the memory areas handled using a given masking object must be of the same kind and have the same size and alignment.

The (optional) hash-based cache table is enabled if and only if **CAESAR_CACHE** is equal to **CAESAR_TRUE**.

If the cache table is disabled, the four next parameters **CAESAR_CACHE_SIZE**, **CAESAR_PRIME**, **CAESAR_COMPARE**, and **CAESAR_HASH** are unused and must be equal to 0, **CAESAR_FALSE**, **NULL**, and **NULL**, respectively; otherwise, the effect of this procedure is undefined. In the sequel, we only consider the case where the cache table is enabled.

Note: If the value of **CAESAR_AREA** is equal to **CAESAR_STRING_AREA_1 ()**, the character strings that will be stored in the masking object pointed to by ***CAESAR_M** may be of arbitrary length. If the cache table is enabled, the contents of these character strings should remain constant, in the sense that, after a character string is passed as argument to the function **CAESAR_APPLY_MASK_1()** defined below, its contents should no longer be modified (as the cache table stores only the string pointer, but not the string contents).

The value of **CAESAR_CACHE_SIZE** determines the maximal number of areas that can be stored simultaneously in the cache table, together with the corresponding characters strings obtained after applying conversion, hiding, and/or renaming operations. Whenever a new area is inserted in the cache table and an existing area is already present with the same hash value as the new area, the new area will replace the existing area. If the value of **CAESAR_CACHE_SIZE** is equal to zero, it is replaced with a default value

greater than zero.

If the value of **CAESAR_PRIME** is equal to **CAESAR_TRUE** and if the value of **CAESAR_CACHE_SIZE** is not a prime number, this value will be replaced by the nearest smaller prime number (since some hash functions require prime modulus). Otherwise, the value of **CAESAR_CACHE_SIZE** will be kept unchanged.

The actual value of the formal parameter **CAESAR_COMPARE** will be stored and associated to the masking object pointed to by ∗**CAESAR_M**. It will be used as a comparison function when it is necessary to decide whether two memory areas are equal or not. If the actual value of **CAESAR_COMPARE** is **NULL**, it may be replaced by a non-**NULL** default value depending on the value of **CAESAR_AREA** and according to the rules specified for function **CAESAR_USE_COMPARE_FUNCTION_AREA_1()** of the ‘‘area_1’’ library. In both cases, all area comparisons will be done using the **CAESAR_COMPARE_AREA_1()** macro defined in the ‘‘area_1’’ library.

The actual value of the formal parameter **CAESAR_HASH** will be stored and associated to the masking object pointed to by ∗**CAESAR_M**. It will be used as a hashing function when it is necessary to compute a hash-value for a memory area. If the actual value of **CAESAR_HASH** is **NULL**, it may be replaced by a non-**NULL** default value depending on the value of **CAESAR_AREA** and according to the rules specified for function **CAESAR_USE_HASH_FUNCTION_AREA_1()** of the ‘‘area_1’’ library. In both cases, area hashing will be done using the **CAESAR_HASH_AREA_1()** macro defined in the ‘‘area_1’’ library.

The actual value of the formal parameter **CAESAR_CONVERT** will be stored and associated to the masking object pointed to by ∗**CAESAR_M**. It will be used as a conversion function when it is necessary to translate a memory area into a character string. If the actual value of **CAESAR_CONVERT** is **NULL**, it may be replaced by a non-**NULL** default value depending on the value of **CAESAR_AREA** and according to the rules specified for function **CAESAR_USE_CONVERT_FUNCTION_AREA_1()** of the ‘‘area_1’’ library. In both cases, all area conversions will be done using the **CAESAR_CONVERT_AREA_1()** macro defined in the ‘‘area_1’’ library.

The value of the formal parameter **CAESAR_HISTORY** must be equal to **CAESAR_TRUE** if the user wants to invoke the **CAESAR_HISTORY_MASK_1()** function defined below.

Note: the list of hiding and renaming operations associated to the masking object pointed to by ∗**CAESAR_M** is not specified at the time the masking object is created but later, using the **CAESAR_USE_HIDE_MASK_1()**, **CAESAR_USE_RENAME_MASK_1()**, and **CAESAR_PARSE_OPTION_MASK_1()** primitives defined below.

.....

CAESAR_DELETE_MASK_1

```
void CAESAR_DELETE_MASK_1 (CAESAR_M)
   CAESAR_TYPE_MASK_1 *CAESAR_M;
   { ... }
```

This procedure frees the memory space corresponding to the masking object pointed to by ∗**CAESAR_M** using **CAESAR_DELETE()**. Afterwards, the **NULL** value is assigned to ∗**CAESAR_M**.

.....

CAESAR_USE_HIDE_MASK_1

```
void CAESAR_USE_HIDE_MASK_1 (CAESAR_M, CAESAR_PATHNAME, CAESAR_KIND,
                             CAESAR_FAILED)
```

```
CAESAR_TYPE_MASK_1 CAESAR_M;
CAESAR_TYPE_STRING CAESAR_PATHNAME;
CAESAR_TYPE_NATURAL CAESAR_KIND;
CAESAR_TYPE_BOOLEAN *CAESAR_FAILED;
{ ... }
```

This function adds a new hiding operation at the end of the (initially empty) list of hiding and renaming operations associated to the masking object pointed to by **CAESAR_M**.

The hiding operation is specified by the actual values of the formal parameters **CAESAR_PATHNAME** and **CAESAR_KIND**, the signification of which is the same as for function **CAESAR_CREATE_HIDE_1()** of the ''hide_1'' library.

Note: the additional formal parameters **CAESAR_POSITIVE_HEADER** and **CAESAR_NEGATIVE_HEADER** of **CAESAR_CREATE_HIDE_1()** are not accessible using **CAESAR_USE_HIDE_MASK_1()**; their default values will be used, meaning that the hiding file referred to by **CAESAR_PATHNAME** must start with either **hide** (positive header) or **hide all but** (negative header).

If the call to **CAESAR_USE_HIDE_MASK_1()** fails, (e.g., because the hiding file is incorrect), the boolean pointed to by ***CAESAR_FAILED** is set to **CAESAR_TRUE**, and an error message is printed to the standard output. Otherwise, the boolean pointed to by ***CAESAR_FAILED** it is set to **CAESAR_FALSE**.

If the cache table is enabled, its contents are erased after any call to **CAESAR_USE_HIDE_MASK_1()**, since the results of previous hiding and/or renaming operations performed so far can no longer be reused after a new hiding operation is added. This allows to invoke **CAESAR_USE_HIDE_MASK_1()** safely, even after a call to the function **CAESAR_APPLY_MASK_1()** defined below.

.....

CAESAR_USE_RENAME_MASK_1

```
void CAESAR_USE_RENAME_MASK_1 (CAESAR_M, CAESAR_PATHNAME, CAESAR_KIND,
                               CAESAR_FAILED)
    CAESAR_TYPE_MASK_1 CAESAR_M;
    CAESAR_TYPE_STRING CAESAR_PATHNAME;
    CAESAR_TYPE_NATURAL CAESAR_KIND;
    CAESAR_TYPE_BOOLEAN *CAESAR_FAILED;
    { ... }
```

This function adds a new renaming operation at the end of the (initially empty) list of hiding and renaming operations associated to the masking object pointed to by **CAESAR_M**.

The renaming operation is specified by the actual values of the formal parameters **CAESAR_PATHNAME** and **CAESAR_KIND**, the signification of which is the same as for function **CAESAR_CREATE_RENAME_1()** of the ''rename_1'' library.

Note: the additional formal parameter **CAESAR_HEADER** of **CAESAR_CREATE_RENAME_1()** is not accessible using **CAESAR_USE_RENAME_MASK_1()**; its default value will be used, meaning that the renaming file referred to by **CAESAR_PATHNAME** must start with **rename**.

If the call to **CAESAR_USE_RENAME_MASK_1()** fails, (e.g., because the renaming file is incorrect), the boolean pointed to by ***CAESAR_FAILED** is set to **CAESAR_TRUE**, and an error message is printed to the standard output. Otherwise, the boolean pointed to by ***CAESAR_FAILED** it is set to **CAESAR_FALSE**.

If the cache table is enabled, its contents are erased after any call to **CAESAR_USE_RENAME_MASK_1()**, since the results of previous hiding and/or renaming operations performed so far can no longer be reused after a new renaming operation is added. This allows to invoke **CAESAR_USE_RENAME_MASK_1()** safely, even after a call to the function **CAESAR_APPLY_MASK_1()** defined below.

.....

CAESAR_PARSE_OPTION_MASK_1

```
void CAESAR_PARSE_OPTION_MASK_1 (CAESAR_M, CAESAR_ARGC, CAESAR_ARGV,
                                 CAESAR_NB_TOKENS)
   CAESAR_TYPE_MASK_1 CAESAR_M;
   CAESAR_TYPE_ARGC CAESAR_ARGC;
   CAESAR_TYPE_ARGV CAESAR_ARGV;
   CAESAR_TYPE_GENUINE_INT *CAESAR_NB_TOKENS;
   { ... }
```

This procedure parses the command-line options specified by parameters **CAESAR_ARGC** and **CAESAR_ARGV** and, if specific options defining a hiding or renaming operation are recognized, the corresponding hiding or renaming operation is added to the masking object pointed to by **CAESAR_M**; this addition is done by invoking the **CAESAR_USE_HIDE_MASK_1()** or **CAESAR_USE_RENAME_MASK_1()** procedure internally.

The parameters **CAESAR_ARGC** and **CAESAR_ARGV** specify a list of character strings (named options). They follow the standard conventions regarding the **argc** and **argv** parameters of the **main()** function of any C program. **CAESAR_ARGC** denotes the number of options and **CAESAR_ARGV** is a character string array containing the options.

A hiding operation is defined by the following sequence of command-line options:

-hide [-total | -partial | -gate] *hiding_filename*

where *hiding_filename* is the pathname of a hiding file (the format of this file is described in the ''hide_1'' library), and where the optional **-total**, **-partial**, and **-gate** options correspond, respectively, to the ''total matching'', ''partial matching'', and ''gate matching'' semantics defined in the documentation of the ''hide_1'' library; option **-total** is the default.

A renaming operation is defined by the following sequence of command-line options:

-rename [-total | -single | -multiple | -gate] *renaming_filename*

where *renaming_filename* is the pathname of a renaming file (the format of this file is described in the ''rename_1'' library), and where the optional **-total**, **-single**, **-multiple**, and **-gate** options correspond, respectively, to the ''total matching'', ''single partial matching'', ''multiple partial matching'', and ''gate matching'' semantics defined in the documentation of the ''rename_1'' library; option **-total** is the default.

The **CAESAR_PARSE_OPTION_MASK_1()** procedure processes command-line options from left to right. One single call to **CAESAR_PARSE_OPTION_MASK_1()** recognizes at most one (hiding or renaming) operation. Thus, **CAESAR_PARSE_OPTION_MASK_1()** should be used within a program loop in which **CAESAR_ARGC** is decremented and **CAESAR_ARGV** is incremented as hiding or renaming options are recognized.

An integer value is assigned to ***CAESAR_NB_TOKENS** in order to indicate how many command-line

options have been recognized and processed. The conventions are the following:

- If ∗**CAESAR_NB_TOKENS** is a strictly positive integer N, the N-th first options have been recognized as forming either a hiding operation or a renaming operation. In this case, **CAESAR_ARGC** will need to be decremented by N and **CAESAR_ARGV** will need to be incremented by N.
- If ∗**CAESAR_NB_TOKENS** is equal to 0, then the first option of **CAESAR_ARGV** has not been recognized as forming either a hiding operation or a renaming operation.
- If ∗**CAESAR_NB_TOKENS** is strictly negative, the first options have been recognized as forming (at least, the beginning of) either a hiding operation or renaming operation, but some error occured while processing these options (e.g., incorrect option syntax, unreadable hiding or renaming file, memory shortage, ...). In this case, an error message will be printed to the standard output. This situation is usually considered as a fatal error.

.....

CAESAR_APPLY_MASK_1

```
CAESAR_TYPE_STRING CAESAR_APPLY_MASK_1 (CAESAR_M, CAESAR_P)
   CAESAR_TYPE_MASK_1 CAESAR_M;
   CAESAR_TYPE_POINTER CAESAR_P;
   { ... }
```

This function first converts the memory area pointed to by **CAESAR_P** into a character string S using the conversion function associated to the masking object pointed to by **CAESAR_M**.

Then, this function applies, in sequence, to S the list of hiding and/or renaming operations associated to the masking object pointed to by **CAESAR_M** and returns the resulting character string.

Note: hiding is done using the **CAESAR_TEST_HIDE_1()** function of the ‘‘hide_1’’ library, a hidden character string being renamed into the constant character string **"i"**.

Note: renaming is done using the **CAESAR_TEST_RENAME_1()** operation of the ‘‘rename_1’’ library.

The order in which hiding and/or renaming operations are applied is the same as the order in which the procedures **CAESAR_USE_HIDE_MASK_1()** and/or **CAESAR_USE_RENAME_MASK_1()** have been called, successively, for the masking object pointed to by **CAESAR_M**. If the list of operations is empty (i.e., if no calls to **CAESAR_USE_HIDE_1()** nor to **CAESAR_USE_RENAME_1()** have occurred), then the result is simply S.

Note: if enabled, the cache table will be used to speed up the execution of **CAESAR_APPLY_MASK_1()**.

Note: if the cache table is enabled and if the masking object was created to handle string areas, meaning that the **CAESAR_AREA** parameter of **CAESAR_CREATE_MASK_1()** was equal to **CAESAR_STRING_AREA_1()**, then ∗**CAESAR_P** is a pointer to a null-terminated character string. The contents of this string must remain constant, i.e., it is forbidden to modify them after invoking of **CAESAR_APPLY_MASK_1{**} until the masking object is deleted.

Note: as regards side effects, the contents of **CAESAR_M** and **CAESAR_P** are not modified, except for the cache table associated to **CAESAR_M**, if any, which might be modified by a call to to **CAESAR_APPLY_MASK_1()**.

Note: it is not allowed to modify the character string returned by **CAESAR_APPLY_MASK_1()** nor to free it, for instance using **free(3)**.

Note: the contents of the character string returned by **CAESAR_APPLY_MASK_1()** may be destroyed by a subsequent call to this function.

.....

CAESAR_LABEL_HIDE_FORMAT_MASK_1

```
#define CAESAR_LABEL_HIDE_FORMAT_MASK_1 "   with labels hidden using \"%s\""
```

This macro-definition returns the default value used for the parameter **CAESAR_HIDE_FORMAT** of function **CAESAR_HISTORY_MASK_1()** when the actual value given to this parameter is **NULL**.

.....

CAESAR_LABEL_RENAME_FORMAT_MASK_1

```
#define CAESAR_LABEL_RENAME_FORMAT_MASK_1 "   with labels renamed using \"%s\""
```

This macro-definition returns the default value used for the parameter **CAESAR_RENAME_FORMAT** of function **CAESAR_HISTORY_MASK_1()** when the actual value given to this parameter is **NULL**.

.....

CAESAR_HISTORY_MASK_1

```
CAESAR_TYPE_STRING CAESAR_HISTORY_MASK_1 (CAESAR_M, CAESAR_PREFIX,
                                          CAESAR_HIDE_FORMAT,
                                          CAESAR_RENAME_FORMAT,
                                          CAESAR_SUFFIX, CAESAR_SEPARATOR)
   CAESAR_TYPE_MASK_1 CAESAR_M;
   CAESAR_TYPE_STRING CAESAR_PREFIX;
   CAESAR_TYPE_STRING CAESAR_HIDE_FORMAT;
   CAESAR_TYPE_STRING CAESAR_RENAME_FORMAT;
   CAESAR_TYPE_STRING CAESAR_SUFFIX;
   CAESAR_TYPE_STRING CAESAR_SEPARATOR;
   { ... }
```

If the masking object pointed to by **CAESAR_M** was created by a call to **CAESAR_CREATE_MASK_1()** in which the value of the **CAESAR_HISTORY** parameter was equal to **CAESAR_FALSE**, the effect of this function is undefined.

Otherwise, this function allocates and returns a character string S summarizing the list of hiding and/or renaming operations associated to the masking object pointed to by **CAESAR_M**. If allocation fails, then S will be equal to **NULL**; if not, S will consist of several components:

- If the character string pointed to by **CAESAR_PREFIX** is different from **NULL**, it will form the first component of S.
- Then, for each hiding or renaming operation associated to the masking object, a corresponding component will be appended to S; these operations are considered in the order in which the **CAESAR_USE_HIDE_MASK_1()** and **CAESAR_USE_RENAME_MASK_1()** procedures have been called.

  For a hiding operation, the corresponding component is given by the character string pointed to by

CAESAR_HIDE_FORMAT. This character string follows the same conventions as a format string given to the **printf(3)** function. However, this string may contain at most one **%s** pattern, which will be substituted by the pathname of the hiding file associated to the operation. It should not contain any other pattern such as **%d**, **%x**, or even **%32s**. If **CAESAR_HIDE_FORMAT** is equal to **NULL**, it will be replaced by its default value **CAESAR_LABEL_HIDE_FORMAT_MASK_1**.

For a renaming operation, the corresponding component is given by the character string pointed to by **CAESAR_RENAME_FORMAT**. This character string follows the same conventions as a format string given to the **printf(3)** function. However, this string may contain at most one **%s** pattern, which will be substituted by the pathname of the renaming file associated to the operation. It should not contain any other pattern such as **%d**, **%x**, or even **%32s**. If **CAESAR_RENAME_FORMAT** is equal to **NULL**, it will be replaced by its default value **CAESAR_LABEL_RENAME_FORMAT_MASK_1**.

- If the character string pointed to by **CAESAR_SUFFIX** is different from **NULL**, it will form the last component of S.
- The character string pointed to by **CAESAR_SEPARATOR** (or, if **CAESAR_SEPARATOR** is **NULL**, the default string **"\n"**) will be inserted as a separator between every two components of S.

Note: The resulting character string S is allocated using **CAESAR_CREATE()** and should be freed as soon as possible using **CAESAR_DELETE()**.

.....

CAESAR_FAILURE_MASK_1

```
void CAESAR_FAILURE_MASK_1 (CAESAR_M)
   CAESAR_TYPE_MASK_1 CAESAR_M;
   { ... }
```

This function returns the value of the failure counter (i.e., the number of searches that failed) of the cache table associated to the masking object pointed to by **CAESAR_M**.

If the cache table is not enabled in the masking object, the effect of this function is undefined.

.....

CAESAR_SUCCESS_MASK_1

```
void CAESAR_SUCCESS_MASK_1 (CAESAR_M)
   CAESAR_TYPE_MASK_1 CAESAR_M;
   { ... }
```

This function returns the value of the success counter (i.e., the number of searches that succeeded) of the cache table associated to the masking object pointed to by **CAESAR_M**.

If the cache table is not enabled in the masking object, the effect of this function is undefined.

.....

CAESAR_FORMAT_MASK_1

```
CAESAR_TYPE_FORMAT CAESAR_FORMAT_MASK_1 (CAESAR_M, CAESAR_FORMAT)
   CAESAR_TYPE_MASK_1 CAESAR_M;
   CAESAR_TYPE_FORMAT CAESAR_FORMAT;
```

```
   { ... }
```

This function allows to control the format under which the masking object pointed to by **CAESAR_M** will be printed by the procedure **CAESAR_PRINT_MASK_1()** (see below). Currently, the following formats are available:

- With format 0, statistical information about the masking object is displayed such as: the area size (in bytes), the area hashable size (in bytes), the area comparison function, the area hash function, the area conversion function, etc. The cache size is also displayed together with, if it is different from zero, the number of searches that failed and succeeded.
- With format 1, the list of operations performed by the masking object is displayed using the procedures **CAESAR_PRINT_HIDE_1()** and **CAESAR_PRINT_RENAME_1()** exported by the ''hide_1'' and ''rename_1'' libraries (these procedures are called with their respective formats set to 0).
- With format 2, the contents of the cache table are displayed if the cache table is enabled.
- (no other format available yet).

By default, the current format of each masking object is initialized to 0.

When called with **CAESAR_FORMAT** between 0 and 2, this fonction sets the current format of **CAESAR_M** to **CAESAR_FORMAT** and returns an undefined result.

When called with another value of **CAESAR_FORMAT**, this function does not modify the current format of **CAESAR_M** but returns a result defined as follows. If **CAESAR_FORMAT** is equal to the constant **CAESAR_CURRENT_FORMAT**, the result is the value of the current format of **CAESAR_M**. If **CAESAR_FORMAT** is equal to the constant **CAESAR_MAXIMAL_FORMAT**, the result is the maximal format value (i.e., 2). In all other cases, the effect of this function is undefined.

.....

CAESAR_MAX_FORMAT_MASK_1

```
CAESAR_TYPE_FORMAT CAESAR_MAX_FORMAT_MASK_1 ()
   { ... }
```

Caution! This function is deprecated. It should no longer be used, as it might be removed from future versions of the *OPEN/CAESAR*. Use function **CAESAR_FORMAT_MASK_1()** instead, called with argument **CAESAR_MAXIMAL_FORMAT**.

This function returns the maximal format value available for printing masking objects.

.....

CAESAR_PRINT_MASK_1

```
void CAESAR_PRINT_MASK_1 (CAESAR_FILE, CAESAR_M)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_MASK_1 CAESAR_M;
   { ... }
```

This procedure prints to file **CAESAR_FILE** a text containing information about the masking object pointed to by **CAESAR_M**. The nature of the information is determined by the current format of the masking object pointed to by **CAESAR_M**.

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

.....

AUTHOR(S)

Nicolas Descoubes and 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