NAME

HTML::Entities − Encode or decode strings with HTML entities

SYNOPSIS

```
use HTML::Entities;

$a = "V&aring;re norske tegn b&oslash;r &#230res";
decode_entities($a);
encode_entities($a, "\200-\377");
```

For example, this:

```
$input = "vis-à-vis Beyoncé's naïve\npapier-mâché résumé";
print encode_entities($input), "\n"
```

Prints this out:

```
vis-&agrave;-vis Beyonc&eacute;'s na&iuml;ve
papier-m&acirc;ch&eacute; r&eacute;sum&eacute;
```

DESCRIPTION

This module deals with encoding and decoding of strings with HTML character entities. The module provides the following functions:

decode_entities(`$string`, ...)

This routine replaces HTML entities found in the `$string` with the corresponding Unicode character. Unrecognized entities are left alone.

If multiple strings are provided as argument they are each decoded separately and the same number of strings are returned.

If called in void context the arguments are decoded in-place.

This routine is exported by default.

_decode_entities(`$string`, \%entity2char)

_decode_entities(`$string`, \%entity2char, `$expand_prefix`)

This will in-place replace HTML entities in `$string`. The`%entity2char` hash must be pro vided. Named entities not found in the `%entity2char` hash are left alone. Numeric entities are expanded unless their value overflow.

The keys in `%entity2char` are the entity names to be expanded and their values are what they should expand into. The values do not have to be single character strings. If a key has ";" as suffix, then occurrences in `$string` are only expanded if properly terminated with ";". Entities without ";" will be expanded regardless of how they are terminated for compatibility with how common browsers treat entities in the Latin-1 range.

If `$expand_prefix` is TRUE then entities without trailing ";" in `%entity2char` will even be expanded as a prefix of a longer unrecognized name. The longest matching name in `%entity2char` will be used. This is mainly present for compatibility with an MSIE misfeature.

```
$string = "foo&nbspbar";
_decode_entities($string, { nb => "@", nbsp => "\xA0" }, 1);
print $string;  # will print "foobar"
```

This routine is exported by default.

encode_entities(`$string`)

encode_entities(`$string`, `$unsafe_chars`)

This routine replaces unsafe characters in `$string` with their entity representation. A second argument can be given to specify which characters to consider unsafe. The unsafe characters is specified using the regular expression character class syntax (what you find within brackets in regular expressions).

The default set of characters to encode are control chars, high-bit chars, and the <, &, >, ' and " characters. But this, for example, would encode *just* the <, &, >, and " characters:

```
$encoded = encode_entities($input, '<>&"');
```

and this would only encode non-plain ASCII:

```
$encoded = encode_entities($input, '^\n\x20-\x25\x27-\x7e');
```

This routine is exported by default.

encode_entities_numeric(`$string`)
encode_entities_numeric(`$string`, `$unsafe_chars`)

This routine works just like encode_entities, except that the replacement entities are always &#x*hexnum*; and never &*entname*;. For example, `encode_entities("r\xF4le")` returns ‘‘rôle’’, but `encode_entities_numeric("r\xF4le")` returns ‘‘rôle’’.

This routine is *not* exported by default. But you can always export it with `use HTML::Entities qw(encode_entities_numeric);` or even `use HTML::Entities qw(:DEFAULT encode_entities_numeric);`

All these routines modify the string passed as the first argument, if called in a void context. In scalar and array contexts, the encoded or decoded string is returned (without changing the input string).

If you prefer not to import these routines into your namespace, you can call them as:

```
use HTML::Entities ();
$decoded = HTML::Entities::decode($a);
$encoded = HTML::Entities::encode($a);
$encoded = HTML::Entities::encode_numeric($a);
```

The module can also export the `%char2entity` and the `%entity2char` hashes, which contain the mapping from all characters to the corresponding entities (and vice versa, respectively).

COPYRIGHT